HA300

SAP HANA Modeling

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

PARTICIPANT HANDBOOK
INSTRUCTOR-LED TRAINING

Course Version: 12
Course Duration: 5 Day(s)
Material Number: 50136340

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

SAP Copyrights and Trademarks

Typographic Conventions

American English is the standard used in this handbook.

The following typographic conventions are also used.

This information is displayed in the instructor's presentation	
Demonstration	
Procedure	
Warning or Caution	
Hint	
Related or Additional Information	
Facilitated Discussion	
User interface control	*Example text*
Window title	*Example text*

Contents

Course Overview

TARGET AUDIENCE

This course is intended for the following audiences:

- Application Consultant
- Data Consultant/Manager
- Database Administrator

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

UNIT 1 Information Views

UNIT OBJECTIVES

- Explain information views
- Describe the main types of information views
- Connect Tables
- Determine the type of join to use when connecting tables
- Create Dimension Calculation views
- Define calculated attributes
- Create time-based dimension views

- Use base table aliases
- Define label columns and hide attributes in a Dimension Calculation View
- Use measures in Calculation Views
- Explain the benefits of each type of node in Calculation Views
- Create and combine nodes in Calculation Views
- Use the features of Calculation Views to enhance the flexibility of this type of view
- Explain Attribute Views and Analytic Views (Deprecated View Types)

Unit 1
Lesson 1

Introducing Information Views

LESSON OVERVIEW

This lesson introduces the general concept of information views, and then describes the different types of views and their benefits.

Business Example

As part of an SAP HANA implementation, you want to use the modeling capabilities of SAP HANA to build flexible information models and easily report on your data.

Before getting into more detail, you want to understand what information views are. This is also an opportunity to review the basic concepts used in reporting; such as dimensions, measures, attributes, hierarchies, and so on.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Explain information views
- Describe the main types of information views

Key Vocabulary of Modeling

Before introducing Modeling in SAP HANA, you must become familiar with some key concepts that are frequently used when reporting on financial or operational data, for example:

- Measure
- Attribute
- Dimension
- Star Schema
- Hierarchy
- Semantics

Measure and Attribute

When you report on data, you have to distinguish the following important concepts:

- Measure
- Attribute

Table 1: Measure vs. Attribute

	Measure	Attribute
Definition	A numeric value, such as a price, quantity, volume, on which you can process arithmetic/statistics operations, such as sum, average, top N values, and calculations.	An element that is used to describe a measure.
Examples	• Number of products sold • Unit Price • Total Price	• Product ID • Product Name • Customer ID • Customer Name • Sales Organization • Sales Org. Country • Sales Org. Region • Currency

Attributes are used to filter or aggregate the measures, in order to answer questions such as:

- What are the total Sales originating from *Sales Org.* located in the *EMEA* region?
- What is the sales revenue generated by the product *Cars*?

Note:
One key objective of modeling in SAP HANA is to create a relevant association between attributes and measures to fulfill a particular reporting requirement.

Dimension

In a number of cases, analyzing the measures is easier if you group attributes together by dimension.

In the examples, the Sales Organization would be treated as a Dimension, with the following associated attributes:

- Country
- Region

Similarly, a Product ID dimension could be associated to several attributes, such as Product Name, Product Category, or Supplier.

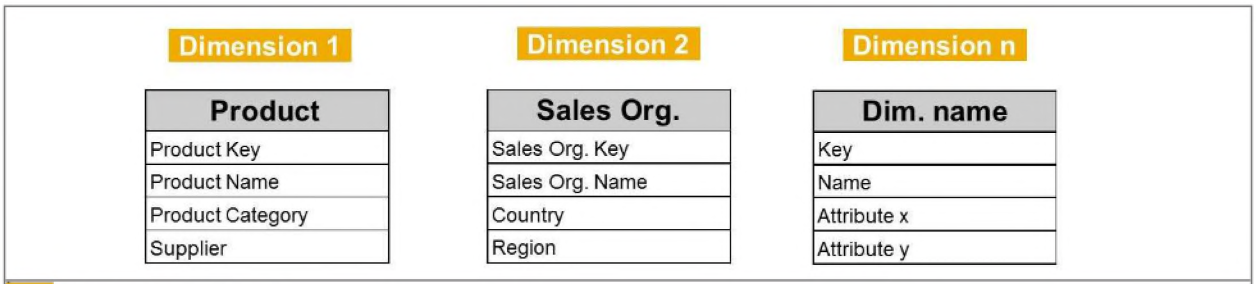


Figure 1: Dimensions

Star Schema

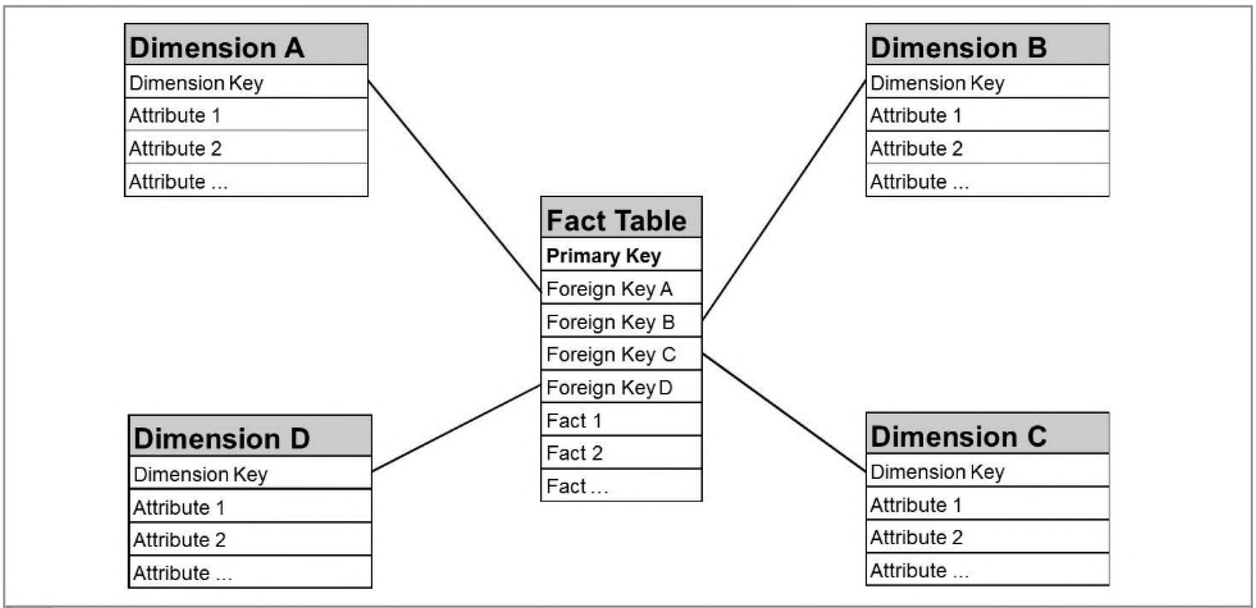


Figure 2: Star Schema

A Star Schema consists of one fact table that references one or several dimensions tables.

The Fact Table contains facts, or measures, as well as the keys used to identify – for each dimension – which dimension member corresponds to each fact.

Each dimension contains attributes, such as the name of the dimension member, description, and other attributes that can be used to filter the dimension members, build a hierarchy, perform specific calculations.

> Note:
> The term *Fact table* is used in a generic way. You will learn later on that Calculation Views in SAP HANA allow you to create this fact table.

Hierarchy

A Hierarchy is a structured representation of an organization, a list of products, the time dimension, and so on, by levels.

It is used to navigate the entire set of members with more ease (the location of the company, the products, or the days, weeks, months, or years) when analyzing the data.

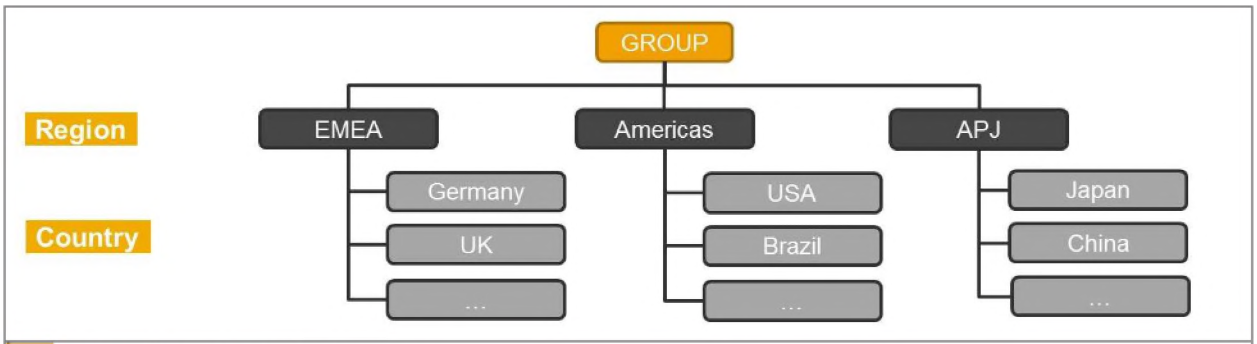


Figure 3: The Hierarchy Concept

In the figure, The Hierarchy Concept, you see a geographical representation of the structure of a company by regions and countries. The level of detail (granularity) of the hierarchy will depend on how the company wants to analyze its data, and its design too. For instance, the geographical hierarchy can be based on where the subsidiaries of a company are located.

In the figure, the top node of the hierarchy, *GROUP*, represents the entire company.

> Note:
> Similarly, the list of products that a company sells could be organized into a hierarchy, by classifying the products by product area, product type.

Semantics

The term Semantics is sometimes used to describe what a data means, or relate to. A numeric data that you report can be of different types. Here are a few examples:

- A monetary value

  For example, the total amount of sales orders.

  In this case, you might need a dimension to specific the currency (for example USD, EUR, or GBP,), if it is not implicit.

- A number of items

  For example, a number of sales orders, or a number of calls to support services.

- A weight, volume, distance, or a compound of these measures

  For example, the payload-distance in freight transportation.

  You often need to specify in which unit the data is expressed.

- A percentage

  For example, a discount rate, or a tax rate.

Note:
Even if the semantics are not always expressed explicitly in the data repository itself, it is very important to know and understand the semantics of any measure to avoid misinterpretation of the data or irrelevant calculation.

For example, before summing up the amount of sales orders, you must make sure they are all expressed in the same currency.

As another example, you have to be careful with aggregations when using columns that contain ratios.

Information Views in SAP HANA

Information Views are used in SAP HANA to create a Virtual Data Model, based on the data that resides in the SAP HANA database schemas.

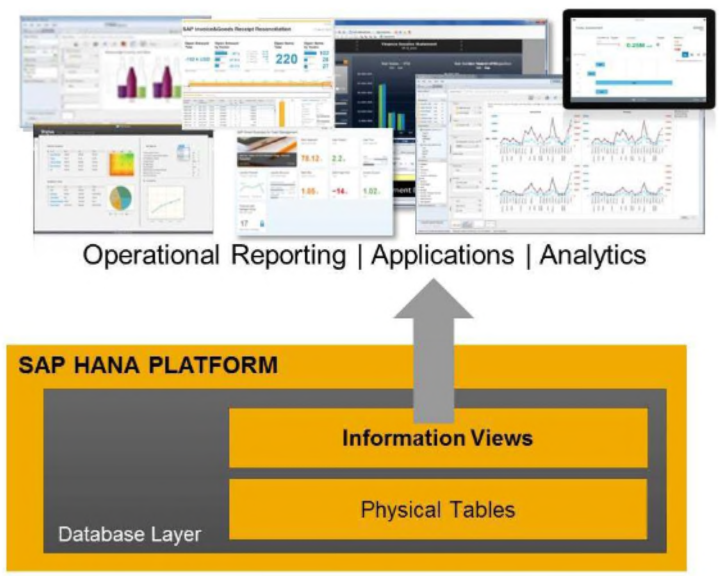


Figure 4: Information Views

The purpose of information views is to organize the data from the individual transactional tables, and to perform a variety of data calculation, in order to get a relevant and meaningful set of measures and dimensions or attributes to answer a specific reporting need. You can make the data more meaningful than in the source tables by customizing the columns names, assigning label columns to key columns, and calculating additional attributes.

Benefits of Information Views

The main benefits of using Information Views in SAP HANA are as follows:

- All calculations are performed on the fly within the database engines

  No aggregates need to be pre-calculated, and the front-end reporting tools delegate most of the data processing workload (filtering, aggregation, calculations) to the SAP HANA in-Memory engines, to achieve very high performance.

- Reusability

  Each Information View can be used (referenced) by other Information Views.

- Flexibility

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

Information Views provide a number of features that make them very flexible. For example, defining hierarchies, filtering data, generating prompts for variables and input parameters, and performing currency conversion.

- Adaptability

  An SAP HANA Information View can adapt its behavior to the list of columns that are selected or projected on top of it. For example, the granularity of a Rank node can be determined dynamically depending on whether you query results by Country or by Country and Customer.

- Easy to transport

  SAP HANA provides powerful tools to transport information models between different SAP HANA databases. For instance, to install SAP-delivered information models, or transport your own information models between your Development, Quality Assurance, and Production landscapes.

> Note:
> These benefits are illustrated in the following lessons of this unit, and in the following units.

Design-Time vs. Runtime Information Views

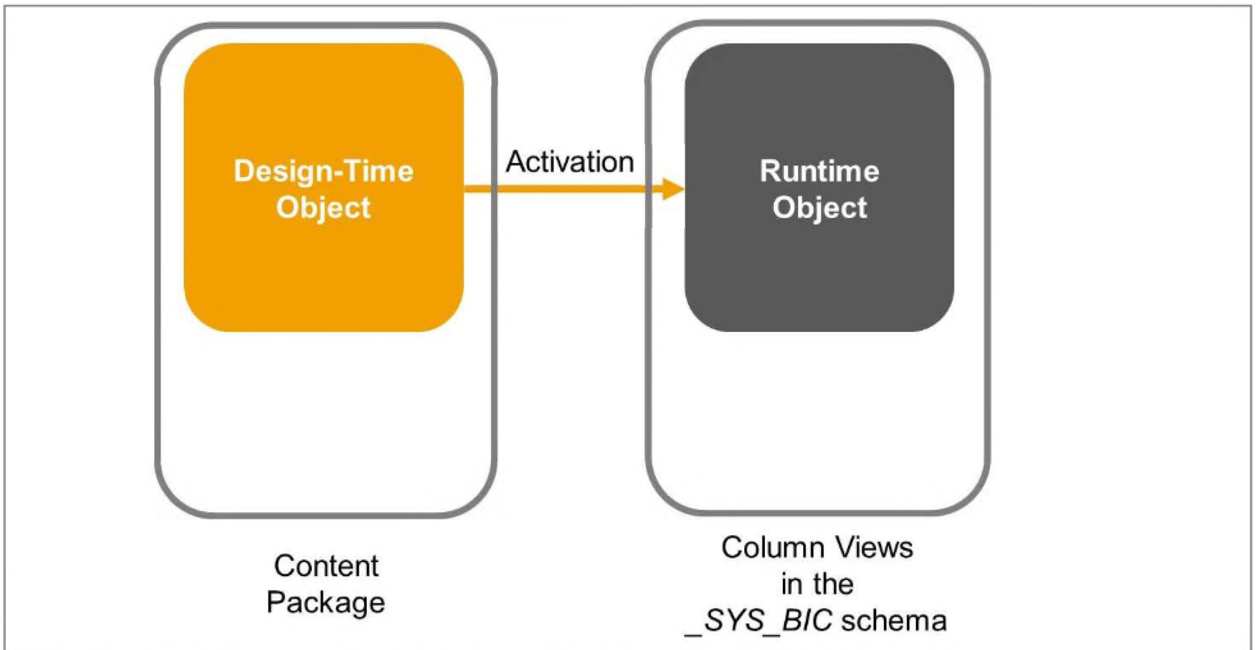


Figure 5: Design Time vs. Runtime Information View

When you create or modify an information view, with SAP HANA Studio or the Web Development Workbench, you work on a Design-Time Object, which is located in a package of the SAP HANA Content. Packages are organized into a hierarchy, and authorizations to access them can be defined at the package level.

The activation of this Design-Time object creates a Runtime Object, which is a Column View located in a dedicated SCHEMA of the SAP HANA Database: the *_SYS_BIC* schema.

Note:
In a number of cases, the activation of a single design-time object creates several related runtime objects in the *_SYS_BIC* schema. For example, in addition to the main column view itself, which contains the measures, you can find a column view with the list of the measures, and column views materializing the hierarchies.

You will learn more about the information models lifecycle later on, in a dedicated unit *Management and Administration of Models*.

Analytical vs. Transactional Requirements

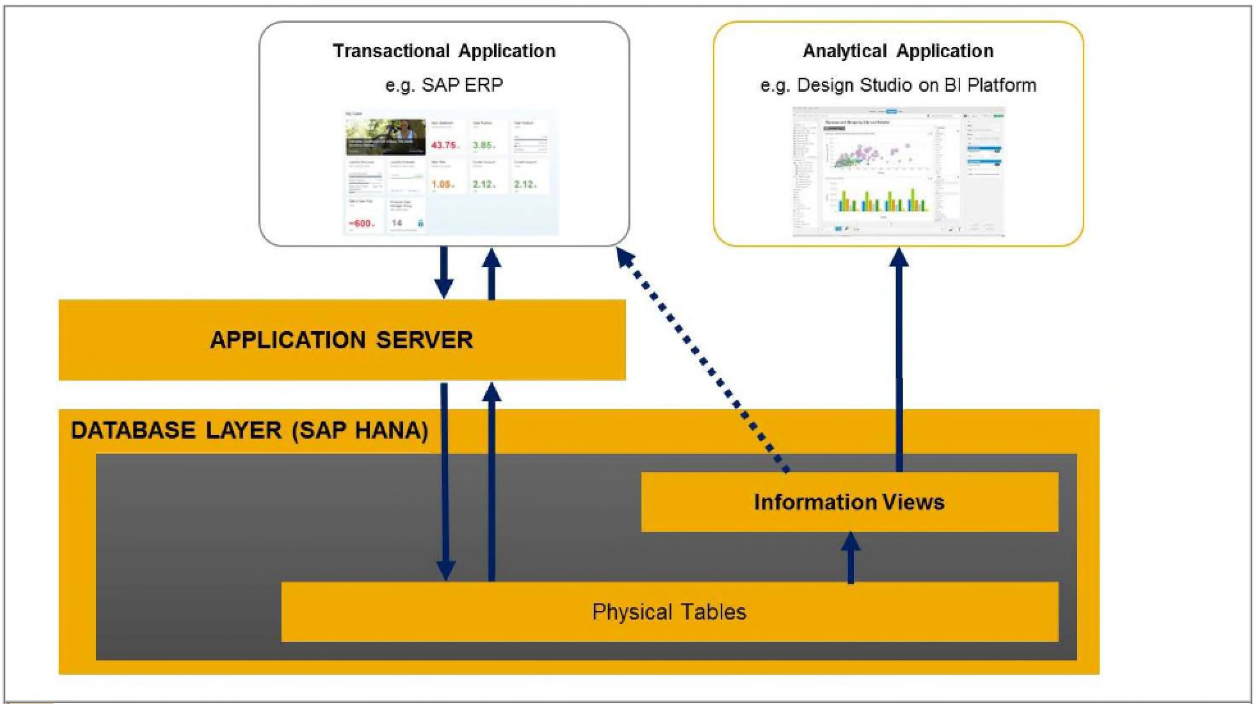


Figure 6: Analytical vs. Transactional Requirements

In transactional applications, such as SAP ERP, the underlying data (stored in physical tables) are generally handled by the application server. This layer is necessary to handle the business process, however complex it can be, and still ensuring data consistency at any time when updating multiple tables, and checking the authorizations of the user.

On the other hand, Information Views are used only to retrieve data, without making any changes to the source transactional data.

For this reason, as shown in the figure, Analytical vs. Transactional Requirements, the analytical process can by-pass the application server, which makes it even faster. The reporting tools query directly the SAP HANA Information Views, where data is calculated on-the-fly.

Note:
In some cases, the transactional application can also leverage the real-time capabilities of SAP HANA Information Views by displaying inside the transactional application (for example in the SAP Fiori Launchpad) real-time analytical data that is relevant for the end user.

Information View Types

There are several types of information views.

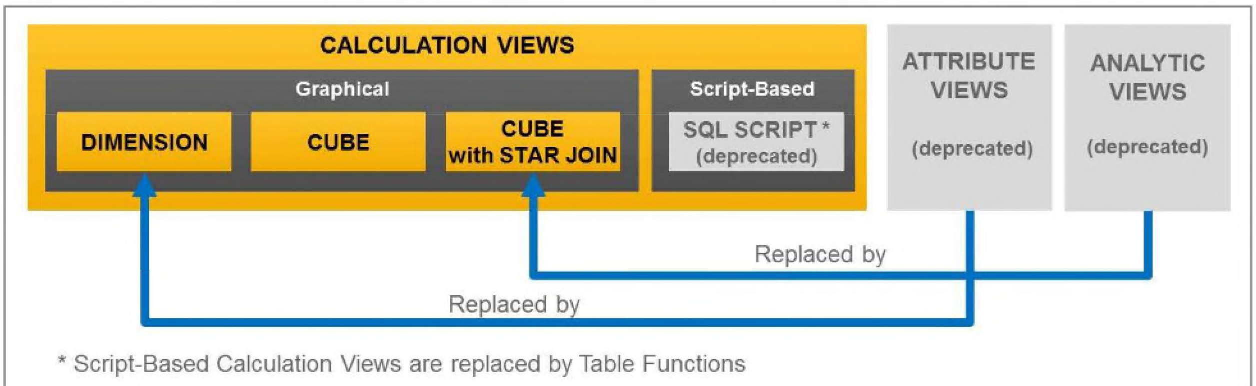


Figure 7: Types of Information Views

With the latest versions of SAP HANA, we recommend that you use Calculation Views exclusively, as they provide the highest level of flexibility.

Another key reason for this is that the enhancements brought to the SQL Optimizer with the latest releases of SAP HANA now provide Calculation Views with the same level of performance as Attribute and Analytic Views.

> Caution:
> Attribute Views and Analytic Views are now deprecated. and a dedicated lesson later on will present these types of views, for the sake of compatibility, as you might still see them in former configurations of SAP HANA.
>
> However, in SAP HANA Service Pack Stacks (SPS) 11, a few features provided by Attribute or Analytic Views are not yet available in Calculation Views. They will also be detailed later on.

Graphical Calculation Views

Before creating Information Views, you must understand the purpose of each type of Graphical Information View.

SAP HANA supports the following main types of Graphical Calculation Views:

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

Calculation View Type	Properties	Default Upper Node
[Blank]	No multidimensional support. Never exposed to any client tool.	Projection
DIMENSION *	No multidimensional support.	Projection
CUBE	Designed for data analysis with multidimensional reporting	Aggregation
CUBE With Star Join	Similar to a *CUBE* Calculation view, but the upper node is a Star Join where you join all the attributes (calculation views of type *DIMENSION* *)	Star Join

* Only *DIMENSION* Calculation View can be joined to the fact table in the *Star Join* node of a *CUBE With Star Join* Calculation View.

Figure 8: Graphical Calculation View Types

In addition to these graphical Calculation Views types, there is a type of Calculation View that does not have an assigned type: the *Data Category* is blank. This type of view is similar to a DIMENSION calculation view, but is never exposed to client tools.

Dimension Calculation Views

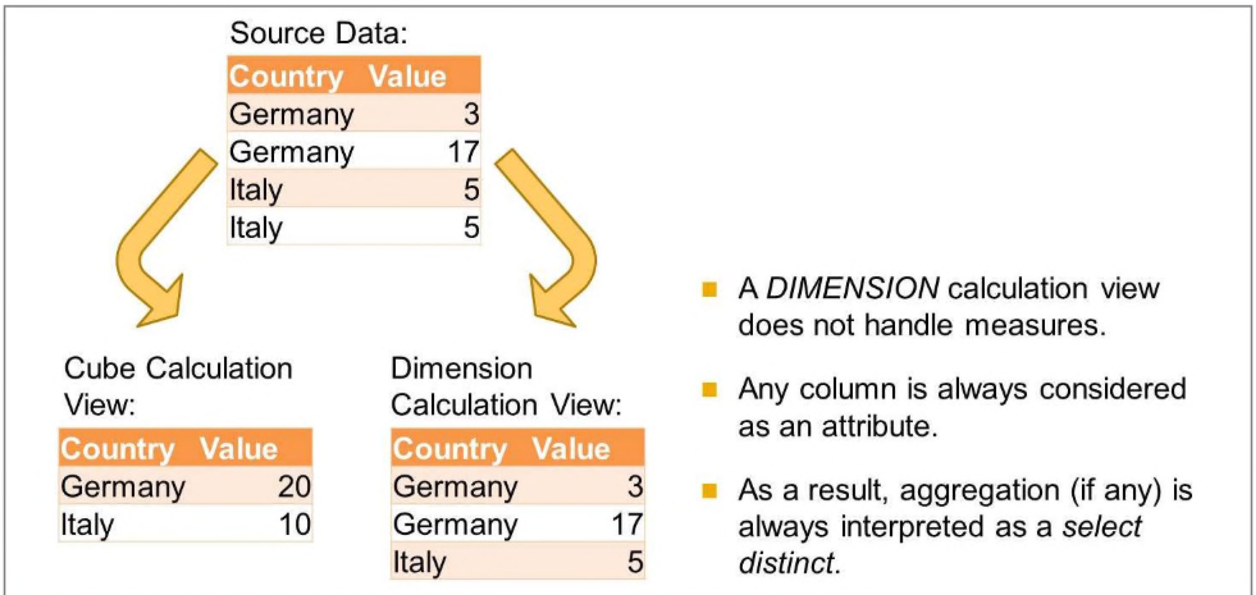


Figure 9: Dimension Calculation Views

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

Note:
Dimension calculation views do not allow measures; any numeric columns or values will always be treated as attributes.

CUBE Calculation Views

When you analyze data including measures, you use a Calculation View of type *CUBE* or *CUBE with Star Join*.

These types of views provide a number of features to filter data, control the aggregation of data, perform complex data calculations on measures, combine data from several data sets, retrieve sub-sets of data based on measure ranking (for example, the top countries for Revenue or Margin).

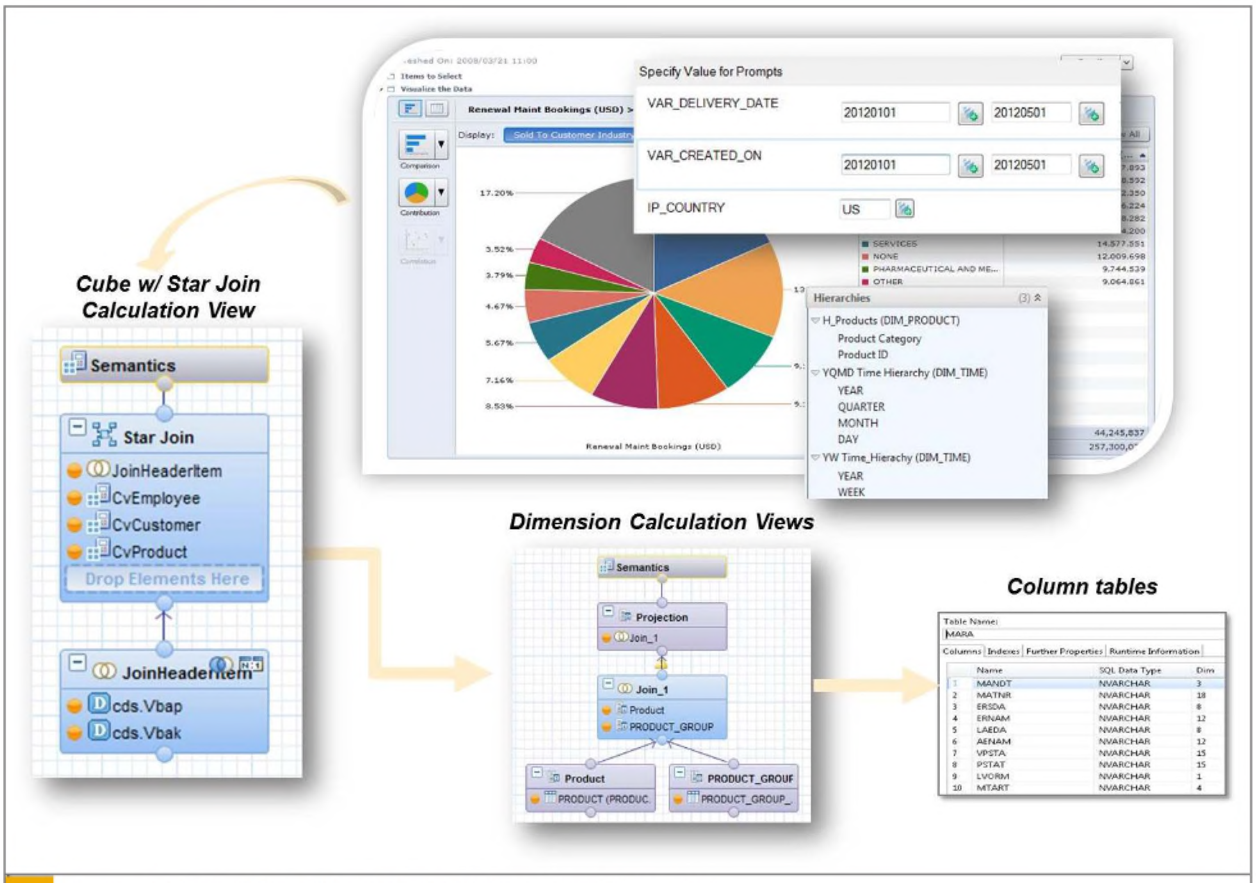


Figure 10: Virtual Data Models for Multidimensional Scenarios

In this scenario, a CUBE with Star Join Calculation View enables a multidimensional reporting that leverages the source data (from the *Vbap* and *Bvak* tables) and dimension Calculation Views created for the main dimensions, such as *Product*, *Employee* or *Customer*.

Multidimensional tools support hierarchies for navigation, filtering and aggregation, as well as prompts (Variables and Input Parameters) for efficient pre-filtering of data.

Virtual Data Models for Normalized Data

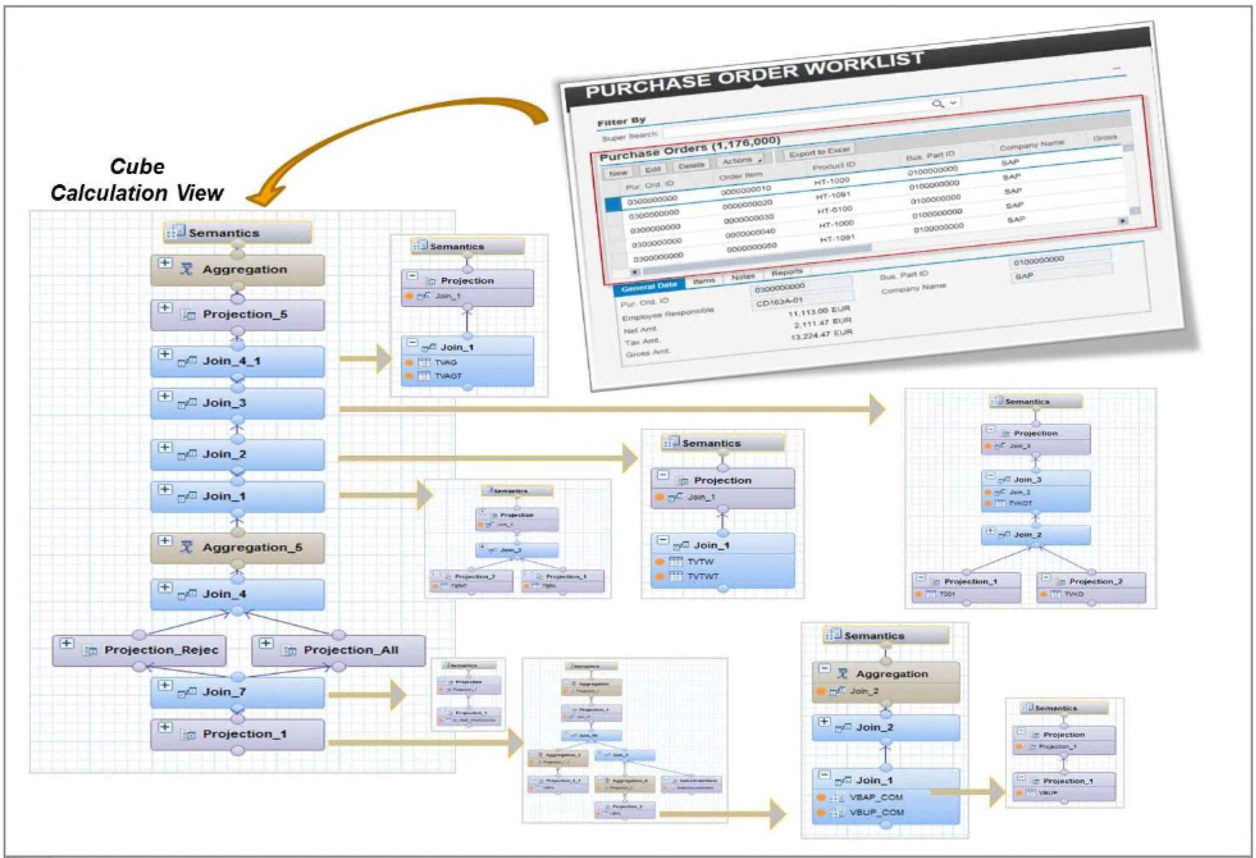


Figure 11: Virtual Data Models for Normalized Data

In this second scenario, SAP HANA Calculation Views typically feed data to Business Applications, such as SAP HANA XS build Applications or Enterprise Analytical Applications. They provide the means to model sophisticated views based on normalized data structures.

To model the Purchase Order Worklist, a Cube Calculation View combines a number of different source views, with Joins, Projections, Aggregations, and so on. Each of the different source views provides a custom set of measures or attributes.

Supported Data Source Types in Graphical Calculation Views

The following is a list of the main data source types that are supported in Calculation Views:

- Row and Column Tables from the same database
- Row and Column Tables from another database of the same SAP HANA System, in a Multi Database Container (MDC) configuration
- Core Data Services (CDS) entities
- SQL Views
- SQL Views from another database of the same SAP HANA System (MDC), from SAP HANA SPS12 onwards
- Graphical Calculation Views

  Note that only Calculation Views of type *DIMENSION* are supported in the *Star Join* node of a Calculation View.
- Calculation Views from another database of the same SAP HANA System (MDC)

Scripted Calculation Views and Graphical Calculation Views embedding an SQLScript Table Function are not supported.

- Table Functions (including Catalog User-Defined Functions from SAP HANA SPS12)
- Decision Tables
- Virtual Tables (data provided through SAP HANA Smart Data Access)
- Analytic Views (deprecated)
- Script-based Calculation Views (deprecated)

Row or Column Table

To identify whether an existing table in SAP HANA is a column table or a row table, you have the following possibilities:

- From the *Systems* view catalog
  - Check the table icon
  - Open the table definition (right-click the table and choose *Open Definition*)
- Within a node in an information view that consumes the table
  - Check the table icon
  - Check the *Properties* view
- From the SQL Console
  - Query the system table, *M_TABLES*

    Example:

```
SELECT "SCHEMA_NAME", "TABLE_NAME", "TABLE_TYPE" FROM "M_TABLES"
WHERE "SCHEMA_NAME" = 'TRAINING'
AND "TABLE_NAME" = 'SALES_DATA'
```

Script-Based Calculation Views

Script-Based Calculation Views can be used when the graphical view types (DIMENSION and CUBE) do not fulfill the reporting requirement.

In particular, they are useful when you need to apply a complex logic that is not supported by graphical information views. For example, when you want to use syntax such as conditions (*IF... THEN... ELSE*), loops (*FOR* or *WHILE*).

> Caution:
> With the last releases of SAP HANA, script-based Calculation Views are now deprecated and replaced by Table Functions.
>
> You will learn more about script-based Calculation Views and Table Functions later in the course, in a dedicated unit *SQL Script and Procedures*. In particular, the additional benefits of Table Functions and the procedure to migrate existing script-based Calculation Views will be covered.

Calculation Views and the SAP HANA Engines

The Index Server of SAP HANA provides several engines to process queries against Calculation Views.

Main SAP HANA Engines to execute Calculation Views

- Join Engine
- OLAP Engine
- Calculation Engine

The distribution of the overall view execution between these engines is complex, and out of the scope of this course.

However, you can keep in mind that optimization processes exist in order to delegate the execution of a Calculation View to the most relevant engine.

Overview of Calculation View Optimization

When executing a query against a Calculation View, SAP HANA uses optimization processes in order to achieve the best possible performance. The optimization processes are enhanced over time, with each release of SAP HANA.

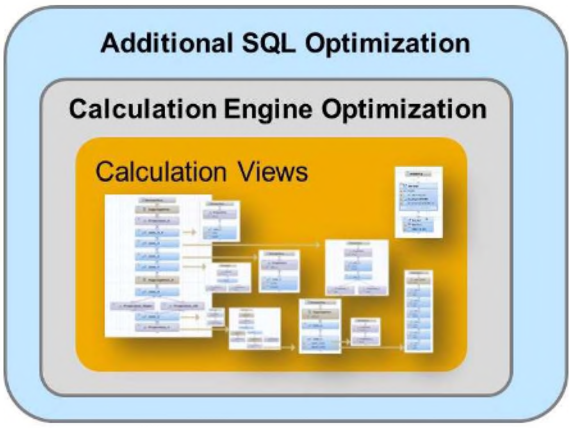


Figure 12: Calculation View Optimization

In SAP HANA SPS10 and above, when you query a Calculation View, the optimization is made in two main steps:

- The initial Calculation Engine optimization generates a single SQL statement across a stacked model, which is then passed to the SQL optimizer.
- The SQL optimizer adds additional optimizations and delegates operations to the best database execution operator.

  For example, it delegates the execution of Star Join to the Online Analytical Processing (OLAP) engine whenever it is possible.

Querying Calculation Views in SAP HANA Studio

When building an Information View, SAP HANA Studio allows you to query the data of the view. It is important to understand the difference between these approaches.

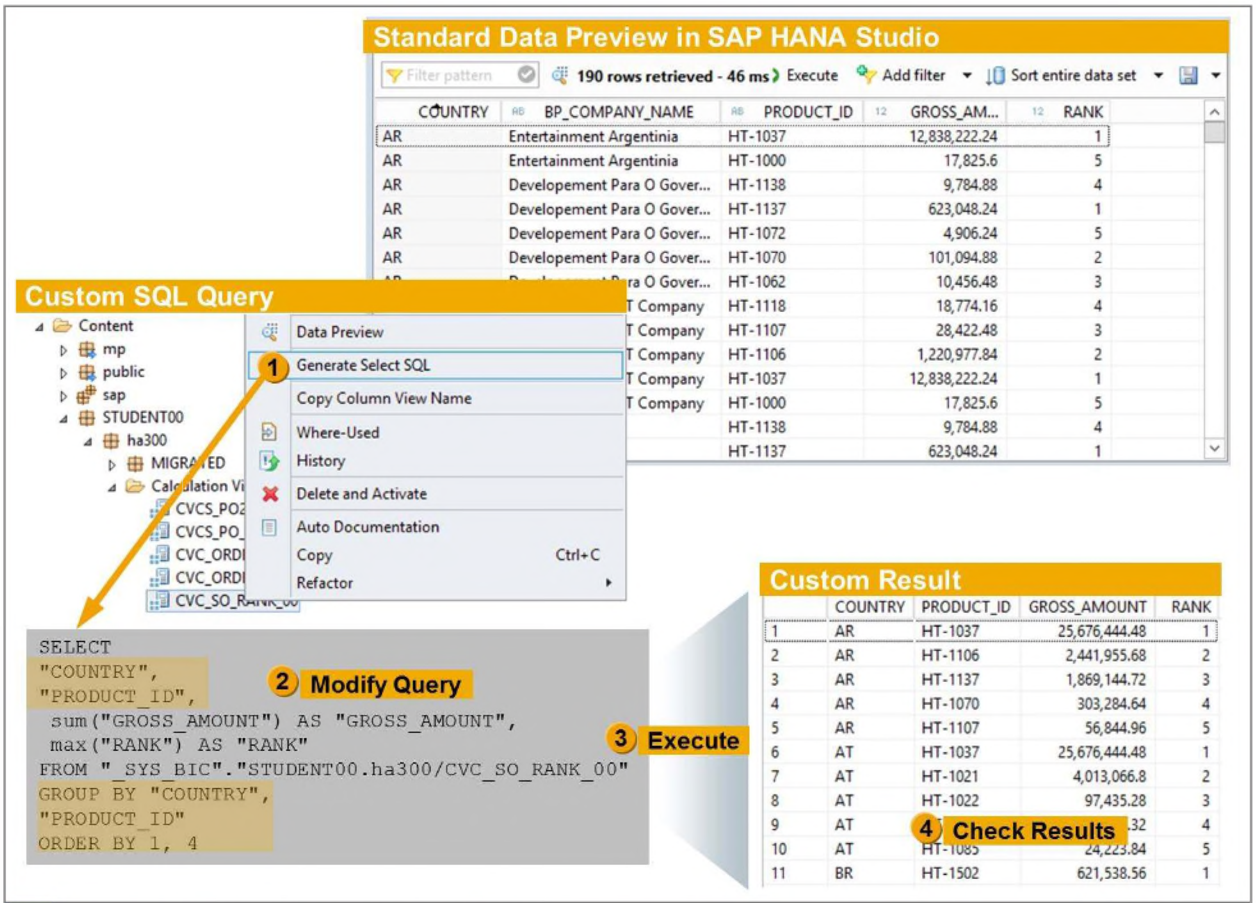


Figure 13: Standard Preview or Custom Query

- Standard Data Preview

  With the Standard Data Preview, you select all the columns that are included in the Semantics of the information view (provided that they are not hidden).

  You can move the columns (drag and drop), apply a temporary filter on one or several attribute columns, and order the result set by one (and only one) column.

- Custom SQL Query

  An alternative to the Standard Data Preview is to execute a Custom SQL Query. As shown in the figure, Standard Preview or Custom Query, after generating the SQL Statement equivalent to the data preview, you can modify it, and for example change the selected columns or the GROUP BY clause, and order the result set by several columns.

  It is particularly useful to perform a thorough test of Calculation Views with a complex scenario (stacked calculation views, counter measures, join between several aggregation nodes with different GROUP BY columns).

> Note:
> Calculation Views behave differently depending on which columns are selected, or whether you explicitly define a group by or not. You must ensure that a View does not give wrong results if it is not correctly queried upon, or document the view so that it is correctly consumed by end-users with reporting applications.

Standard Data Preview Features

Even though the Data Preview Editor is not a reporting tool, it still offers analysis functionality that can be useful during modeling or troubleshooting. It is comprised of the following tabs, each offering specific capabilities:

Table 2: Data Preview Tabs

Tab Page	Displays	Use Case
Raw Data	All data	Basic display of contents
Distinct Values	All Attributes	Basic profiling
Analysis	Selected attributes and measures in tables or graphs	Profiling and analysis

A setting in SAP HANA Studio preferences defines whether the default query must be executed immediately when you display the *Raw Data* tab. This setting is available since SPS10, and is located under *SAP HANA* → *Modeler* → *Data Preview*.

Differing the default query execution can be useful when the user wants to apply additional criteria to the data preview, such as:

- Define the maximum number of rows to return
- Apply filters to one or several columns

  It is also possible to specify whether sorting preview data by a column should be applied to the data set currently displayed in the preview, or to the entire data set.

Suppose you are testing a very complex view, and –just temporarily– want to reduce the result set to one specific Order ID. Executing the default query in that case is not useful, and might make you lose time. Instead you can:

1. In SAP HANA Preferences, deselect the *Execute the default query on data preview* preference.
2. After choosing *Preview Data* for your information model, choose the *Raw Data* tab (the default query will not be executed).
3. Define a filter on the *ORDERID* column.
4. Click *Execute*.

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

Unit 1
Exercise 1

Prepare Your SAP HANA System

Exercise Objectives

After completing this exercise, you will be able to:

- Start SAP HANA Studio
- Connect to the SAP HANA system
- Prepare your system for modeling activities

Business Example

You are at a customer site and want to start the SAP HANA studio to connect to the SAP HANA customer database.

> Note:
> In this exercise, when the values include ##, replace the characters with the number that your instructor assigned you.
>
> For SAP Live Access users, choose the group number 01.

Task 1: Connect to the Training System and Copy Course Files

1. Log on to the Common Training infrastructure.

   > Note:
   > The logon procedure to the Common Training infrastructure will be provided by your instructor or SAP Live Access instructions.

2. Create a remote desktop connection to the training landscape using the following data:

Field	Value
Remote Desktop Alias (Server)	<provided by your instructor or mentioned in your SAP Live Access instructions>
Remote Desktop User Name	`train-##`
Remote Desktop Password	`initial`

   If the following dialog box displays *The identity of the remote computer cannot be verified...*, select the checkbox *Don't ask me again* and choose *Yes*.

3. Add shortcuts to your Windows *Start* screen for the following applications:

- SAP HANA Studio
- Excel 2013
- Analysis for Microsoft Excel

> Hint:
> You can add the shortcuts (pin to *Start* page) from the Windows *Apps* page, or use the search feature (type directly the application or folder name) from the Windows *Start* page.

4. Copy the course files to your local system.

   All the files needed for the course are located in a shared folder *\\kpstransfer\hana\sps12\student* that you can access with a shortcut provided in your Windows system: *student - hanastudent - hanareadonly*. The credentials to this shared folder are the following:

Field	Value
User name	**hanastudent**
Password	**hanareadonly**

   To avoid accessing this shared folder each time you need a file, create a dedicated *student* folder in the N:\ drive of your training system, and copy the entire *HA300* folder to *N:\student*:

   After that, whenever your training material refers to files from the *student* folder, you can find them in your *N:\student\HA300* folder.

5. Add the folder *N:\student\HA300* to your favorite folders in *Windows Explorer*. This allows access to the folder directly from *Windows Explorer*, without expanding the *N:* drive.

Task 2: Start SAP HANA Studio and Define SAP HANA Studio Preferences

1. Start SAP HANA Studio and accept the default locations proposed in the two following dialog boxes:
   - StartUp Tool - HANA Studio: *N:\SAP HANA Studio*
   - Workspace Launcher: *N:\SAP HANA Studio\hdbstudio* (select the checkbox *Set this as the default and do not ask again)*
2. Open the *Modeler* perspective.

   If a *Secure Storage* dialog box opens to provide additional information for password recovery, choose *No*.
3. Maximize the *SAP HANA studio* window and close the *Quick View* pane.
4. Change the network connection preferences.

   Choose *Direct* for the *Active Provider*.
5. Define Internet Explorer as the default web browser.

   In the *SAP HANA Studio Preferences* tree, this setting is found under *General* → *Web Browser*.

6. Activate all the Validation Rules of the Modeler, except the validation rule *Large object data type Check Rule*.

   This rule is located in the *SAP HANA studio Preferences* tree *SAP HANA → Modeler → Validation Rules → View General*.

Task 3: Connect to the SAP HANA System

1. Add the SAP HANA system.

Field	Value
SAP HANA Database Host	**wdflbmt7215.wdf.sap.corp**
SAP HANA Database Instance	**00**
SAP HANA Database Description	SAP HANA

> Note:
> Your training system is configured in *Single Container* mode.

2. Log on to the SAP HANA system with the following credentials.

Field	Value
SAP HANA Database User Name	**STUDENT##** (Replace ## with your group number)
SAP HANA Database Password	**Training1**

Task 4: Prepare your System for Modeling Activities

1. Create a content package in the *SAP HANA Modeler* perspective.

Field	Value
Name	**STUDENT##**
Description	**STUDENT##**
Person Responsible	**STUDENT##**

Unit 1
Solution 1

Prepare Your SAP HANA System

Exercise Objectives

After completing this exercise, you will be able to:

- Start SAP HANA Studio
- Connect to the SAP HANA system
- Prepare your system for modeling activities

Business Example

You are at a customer site and want to start the SAP HANA studio to connect to the SAP HANA customer database.

> Note:
> In this exercise, when the values include ##, replace the characters with the number that your instructor assigned you.
>
> For SAP Live Access users, choose the group number 01.

Task 1: Connect to the Training System and Copy Course Files

1. Log on to the Common Training infrastructure.

> Note:
> The logon procedure to the Common Training infrastructure will be provided by your instructor or SAP Live Access instructions.

2. Create a remote desktop connection to the training landscape using the following data:

Field	Value
Remote Desktop Alias (Server)	<provided by your instructor or mentioned in your SAP Live Access instructions>
Remote Desktop User Name	**train-##**
Remote Desktop Password	**initial**

If the following dialog box displays *The identity of the remote computer cannot be verified...*, select the checkbox *Don't ask me again* and choose *Yes*.

a) On the Windows desktop, double-click the *Remote Desktop Connection* icon.

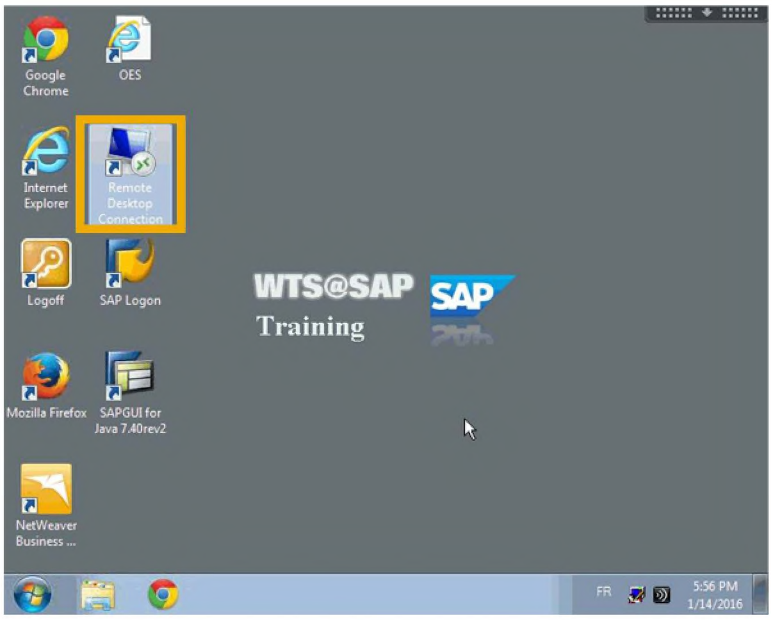


Figure 14: Start Remote Desktop Connection

b) In the *Remote Desktop Connection* dialog box, in the *Computer* field, enter **`Remote Desktop Alias`** and choose *Connect*.



Figure 15: Enter Remote Desktop Alias for Server Connection

c) In the *Windows Security* dialog box, choose *Use another account*.

Enter your remote desktop user name and password.

Choose *OK*.

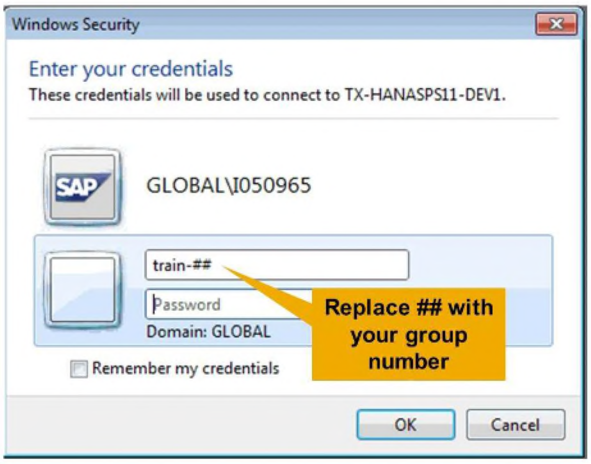


Figure 16: Log On to Remote Desktop Windows Server

3. Add shortcuts to your Windows *Start* screen for the following applications:

   - SAP HANA Studio
   - Excel 2013
   - Analysis for Microsoft Excel

Hint:
You can add the shortcuts (pin to *Start* page) from the Windows *Apps* page, or use the search feature (type directly the application or folder name) from the Windows *Start* page.

a) On the Windows desktop, choose *Start*.

b) Choose the *Down* arrow.

c) Right-click the *SAP HANA Studio* icon and choose *Pin to Start*.

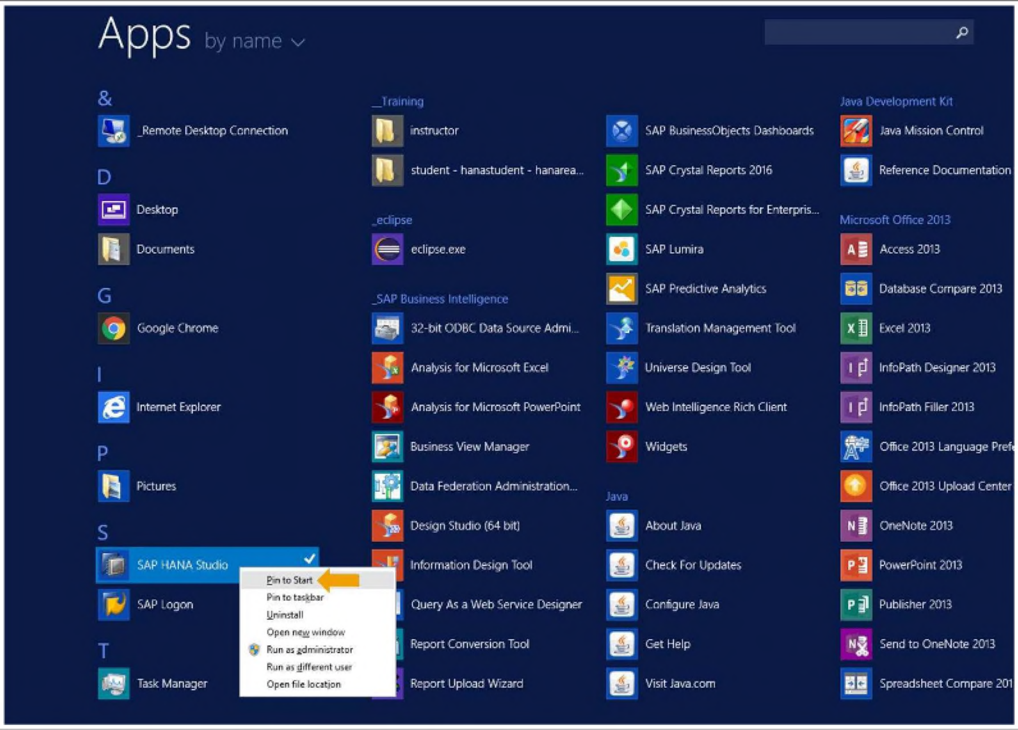


Figure 17: Customizing the Windows Start Screen

d) If necessary, repeat the same steps for the other shortcuts.

4. Copy the course files to your local system.

   All the files needed for the course are located in a shared folder *\\kpstransfer\hana\sps12\student* that you can access with a shortcut provided in your Windows system: *student - hanastudent - hanareadonly*. The credentials to this shared folder are the following:

Field	Value
User name	**hanastudent**
Password	**hanareadonly**

   To avoid accessing this shared folder each time you need a file, create a dedicated *student* folder in the N:\ drive of your training system, and copy the entire *HA300* folder to *N:\student*:

   After that, whenever your training material refers to files from the *student* folder, you can find them in your *N:\student\HA300* folder.

   a) On the Windows desktop, choose *Start*.

   b) Enter **student** and, in the search results, right-click the *student - hanastudent - hanareadonly* shortcut and choose *Open File Location*.

   c) Double-click the *student - hanastudent - hanareadonly* shortcut located in the *__Training* folder.

   d) In the *Windows Explorer* window that opens, right-click the course folder *HA300* and choose *Copy*.

   e) If needed, provide the credentials as per the table.

f) On the left pane of the same *Windows Explorer* window, expand the *wdflbmt7096* item. This is your “local” computer.

g) Right-click the *Application (N:)* drive and choose *New* → *Folder*.

h) Enter the folder name **student** and press Enter.

i) Right-click the new *student* folder and choose *Paste*.

j) Check that you now have an *N:\student\HA300* folder with several files inside.

5. Add the folder *N:\student\HA300* to your favorite folders in *Windows Explorer*. This allows access to the folder directly from *Windows Explorer*, without expanding the *N:* drive.

   a) In *Windows Explorer*, display the content of the *N:\student\HA300* folder.

   b) In the left pane of *Windows Explorer*, right-click the *Favorites* folder and choose *Add current location to Favorites*.

Task 2: Start SAP HANA Studio and Define SAP HANA Studio Preferences

1. Start SAP HANA Studio and accept the default locations proposed in the two following dialog boxes:

   - StartUp Tool - HANA Studio: *N:\SAP HANA Studio*
   - Workspace Launcher: *N:\SAP HANA Studio\hdbstudio* (select the checkbox *Set this as the default and do not ask again)*

   a) In the *Windows Start* screen, double-click *SAP HANA Studio*.

   b) On the *StartUp Tool - SAP HANA Studio* screen, choose *OK*.

   c) In the *Workspace Launcher* screen, select the checkbox *Use this as the default and do not ask again*, and choose *OK*.

2. Open the *Modeler* perspective.

   If a *Secure Storage* dialog box opens to provide additional information for password recovery, choose *No*.

   a) On the *Welcome* screen, choose *Open Modeler*.

3. Maximize the *SAP HANA studio* window and close the *Quick View* pane.

   a) Choose the top-right *Maximise* button.

   b) Choose the *Close* (X) button at the top of the *Quick View* pane.

4. Change the network connection preferences.

   Choose *Direct* for the *Active Provider*.

   a) Choose *Window* → *Preferences* and in the Preferences tree navigate to *General* → *Network Connections*.

   b) Set the *Active Provider* to *Direct*, to open all connections without the use of a proxy server.

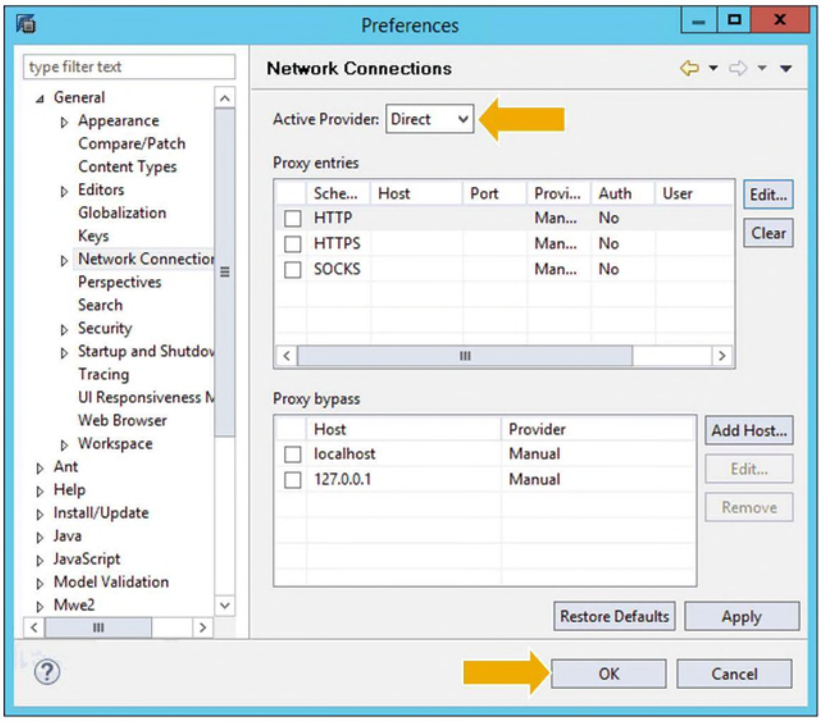


Figure 18: Studio Preference – Network Connections

5. Define Internet Explorer as the default web browser.

   In the *SAP HANA Studio Preferences* tree, this setting is found under *General* → *Web Browser*.

   a) In the Preference tree, navigate to *General* → *Web Browser*.

   b) Select the *Use external web browser* radio button.

   c) Select the *Internet Explorer* check box.

6. Activate all the Validation Rules of the Modeler, except the validation rule *Large object data type Check Rule*.

   This rule is located in the *SAP HANA studio Preferences* tree *SAP HANA* → *Modeler* → *Validation Rules* → *View General*.

   a) Navigate to the Preferences tree to *SAP HANA* → *Modeler* → *Validation Rules*.

   b) Scroll down to the bottom of the *Validation Rules* pane and choose *Check all*.

   c) In *SAP HANA* → *Modeler* → *Validation Rules* → *View General* node, deselect the *Large object data type Check Rule* checkbox.

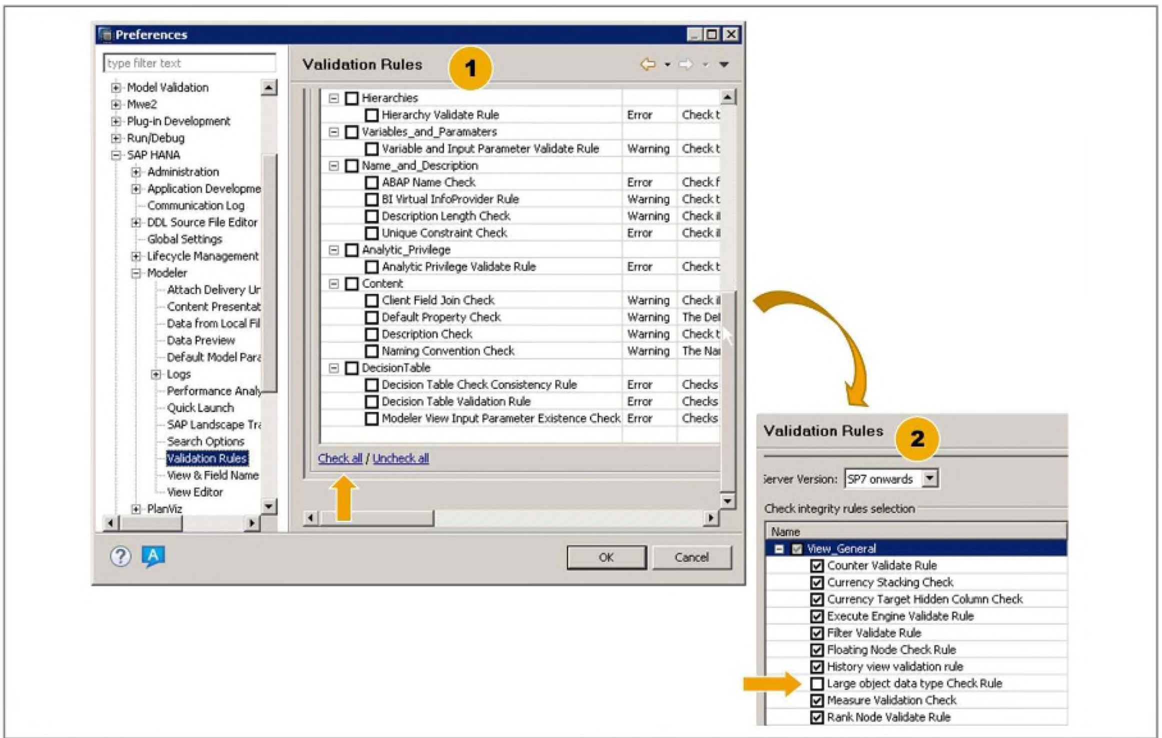


Figure 19: Studio Preference – Activate Validation Rules

d) Choose *OK*.

Task 3: Connect to the SAP HANA System

1. Add the SAP HANA system.

Field	Value
SAP HANA Database Host	**wdflbmt7215.wdf.sap.corp**
SAP HANA Database Instance	**00**
SAP HANA Database Description	SAP HANA

> Note:
> Your training system is configured in *Single Container* mode.

a) To register a new system, right-click anywhere in the *SAP HANA Systems* view, and choose *Add System*.

> Hint:
> Alternatively, choose the *(Add Systems)* button on the top of the *Systems* view and choose *Add System*.

b) Enter the details from the table and choose *Next*.

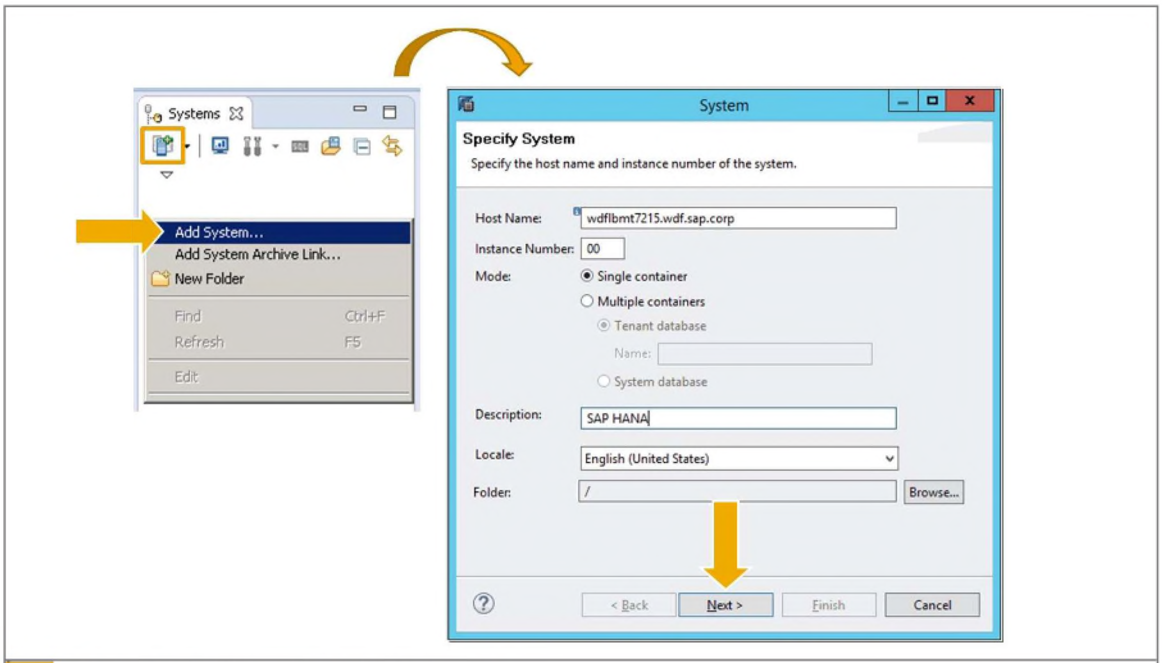


Figure 20: Adding an SAP HANA System

2. Log on to the SAP HANA system with the following credentials.

Field	Value
SAP HANA Database User Name	**STUDENT##** (Replace ## with your group number)
SAP HANA Database Password	**Training1**

a) Enter the credentials as per the table above, and choose *Finish*.

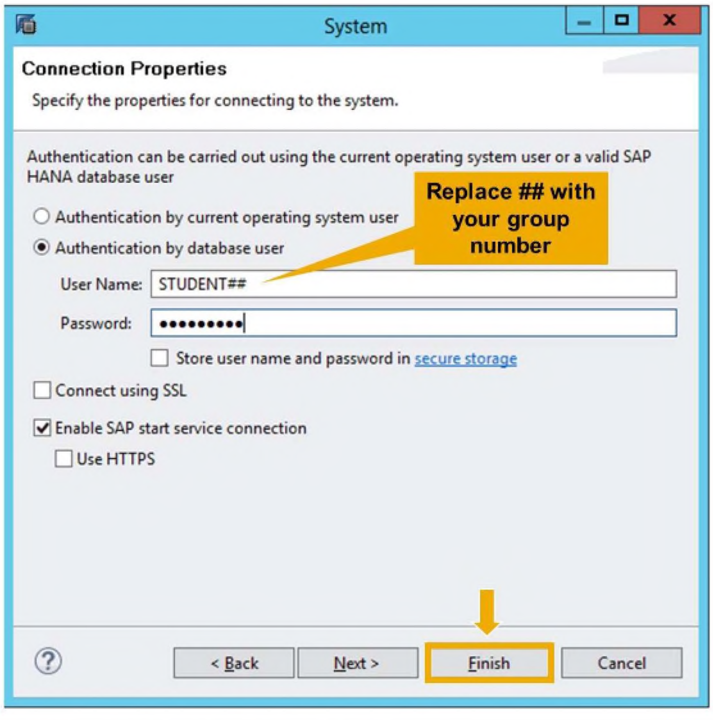


Figure 21: Logging On to the SAP HANA System

Task 4: Prepare your System for Modeling Activities

1. Create a content package in the *SAP HANA Modeler* perspective.

Field	Value
Name	**STUDENT##**
Description	**STUDENT##**
Person Responsible	**STUDENT##**

   a) Switch to the *SAP HANA Modeler* perspective, choose *Window* → *Open Perspective* → *Other* → *SAP HANA Modeler*.

   Choose *OK*.

   b) In the *Systems* view, right-click the *Content* folder and choose *New* → *Package*.

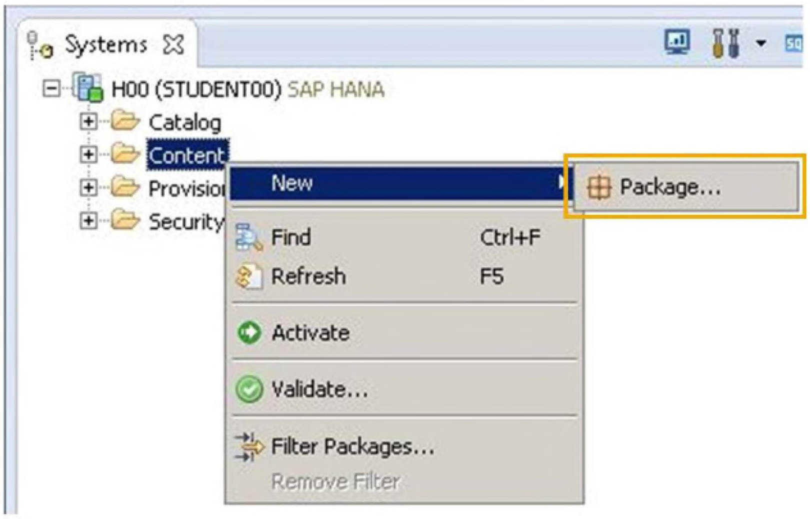


Figure 22: Create New Content Package

   c) Enter the package details as described, and click *OK*.

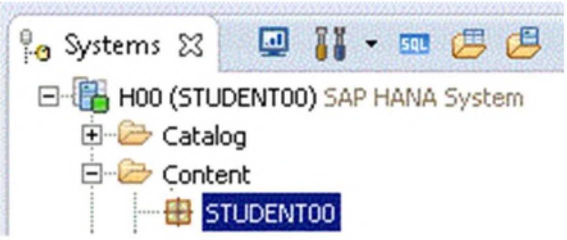


Figure 23: Empty Content Package

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

LESSON SUMMARY

You should now be able to:

- Explain information views
- Describe the main types of information views

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

Unit 1
Lesson 2

Connecting Tables

LESSON OVERVIEW

This lesson describes the different join types that are available in SAP HANA and explains how to connect tables.

In addition to the classical Inner, Left or Right Outer Joins, and Full Outer Join, SAP HANA offers other types of joins that can provide better execution speed or address some specific business scenarios. These SAP HANA specific joins include the following:

- Referential join
- Text join
- Temporal join
- Star join
- Spatial join

Business Example

SAP HANA system host an SAP ERP system, in which the data model is highly normalized, which means that to avoid as much as possible data redundancy, the data is distributed in a huge number of tables. For example, the table containing sales orders does not contain customer addresses or cities. The information about customer is located in a set of different tables. The same applies the product details, and so on.

To retrieve your sales order data based on the name and location of the customer, or the product category, you need to join several tables.

You want to know more about the different types of join, and how they behave in SAP HANA Information Models.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Connect Tables
- Determine the type of join to use when connecting tables

Connecting Tables

One important activity when you create information models is to express the relationships between the different data sources used by a model. Most often, this is done by using joins, a classical artefact in any Relational Database Management System (RDBMS).

To illustrate the behavior of the different types of joins in SAP HANA, consider the following tables:

- Sales Order

- Customer
- State

The objective is to join these tables in order to retrieve the Sales Order amounts (facts) with the customer information, including the states in which the customers reside.

Sales Order

	ORDER_ID	C_ID	AMOUNT
1	1	1	100
2	2	1	100
3	3	2	100
4	4	4	100
5	8	77	100

Customer

	C_ID	CNAME	STATE	AGE
1	1	WERNER	MI	10
2	2	MARK	MI	11
3	3	TOM	TX	12
4	4	BOB	TX	13

We want to connect the ***Sales Order*** **table** to the ***Customer*** **table** linked to **the *State* table.**

State

	STATE	SNAME
1	MI	MICHIGAN
2	AL	ALABAMA

Figure 24: Sample Data for Business Example

As a preliminary, you can make the following observations:

- Sales Order 8 does not have a customer master record.
- Customer TOM does not have any orders.
- State TX does not have a description.
- No customer resides in Alabama.

Join Type Summary

Join type	Use when you want to report on	Be aware that
INNER	Facts with matching dimensions only	* Facts without a dimension will be excluded * Dimensions without a fact will be excluded * Join is always executed
LEFT OUTER	All posted facts whether there is a matching dimension or not	* Dimensions without a fact will be excluded * Best performance since join is omissible
RIGHT OUTER	All dimension whether there are matching facts or not	* Facts without a dimension will be excluded * Right outer join is rarely used
FULL OUTER	All posted facts and all dimensions	* Combines the effects of LEFT and RIGHT OUTER join
REFERENTIAL	Facts with matching dimensions only where referential integrity is ensured	* The join is only executed if columns from the right table are queried
TEXT	A multi-language table	* Requires a language column (SPRAS or equivalent) * Acts as a LEFT OUTER JOIN
TEMPORAL	A key date within a validity period	* Needs a temporal condition, from and to date.
SPATIAL	Geospatial data	* Executes specific computations on spatial data

Figure 25: Join Type Summary

There are different types of joins. The inner and outer joins are similar to other database management systems. Others are specific to SAP HANA.

You can review the typical use cases for the different join types.

> Note:
> This table builds on a scenario similar to the one just described *(Sample Data for Business Example)*, assuming that a fact table (considered as the LEFT table) is joined to a dimension table (considered as the RIGHT table).

Each join type will now be presented in detail.

Inner Join

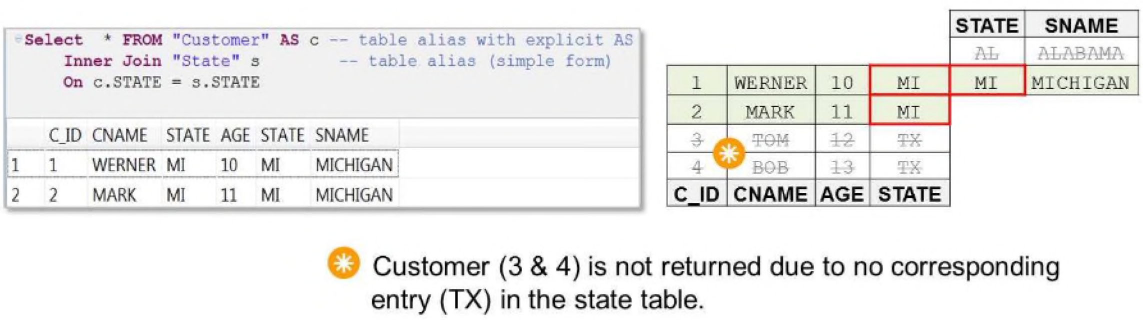


Figure 26: Inner Join in a Dimension Calculation View

The inner join is the most basic of the join types. It returns rows when there is at least one match in both sides of the join.

Inner Join in a Cube Calculation View

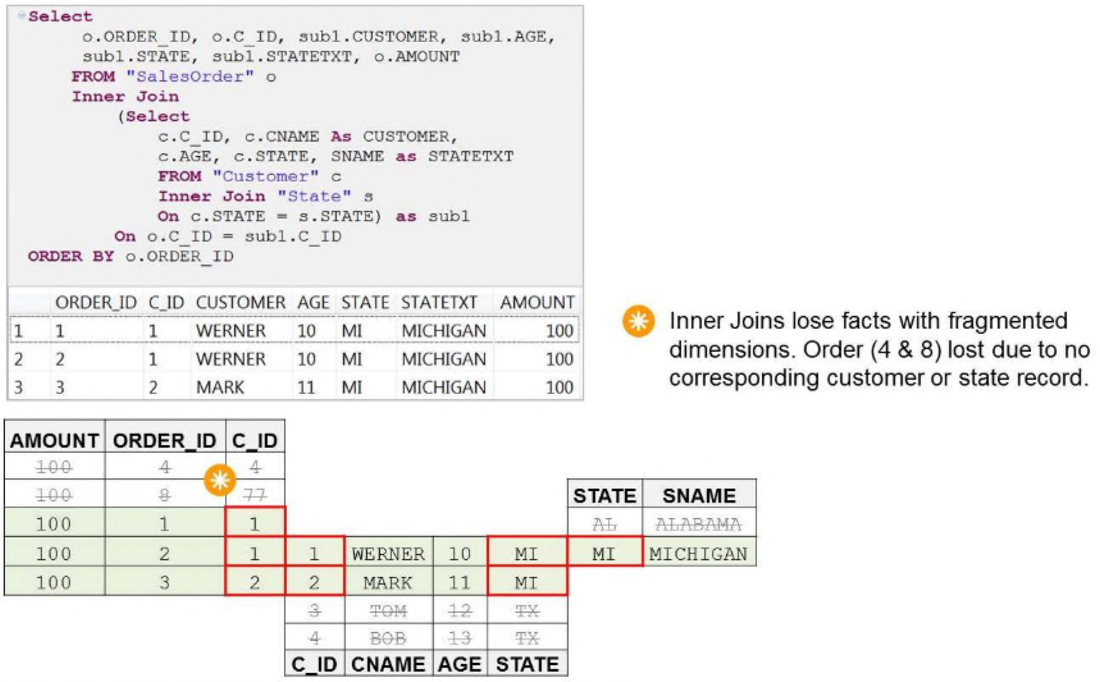


Figure 27: Inner Join in a Cube Calculation View

The figure, Inner Join in a Cube Calculation View, shows the behavior of inner joins in a CUBE Calculation View.

With the sample scenario data, some facts are not retrieved because customer information is missing.

Left Outer Join

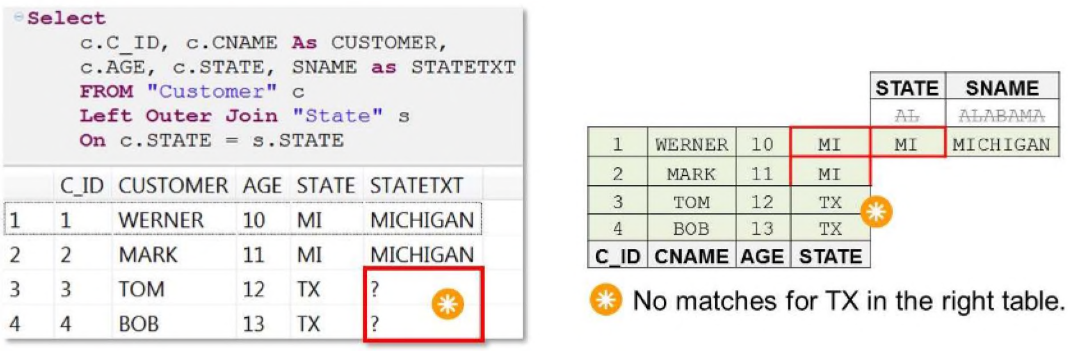


Figure 28: Left Outer Join in a DIMENSION Calculation View

Left Outer Join returns all rows from the left table, even if there are no matches in the right table.

Left Outer Joins and Design Time Filters

CUSTOMER

C_ID	CNAME	STATE	AGE
1	WERNER	MI	10
2	MARK	MI	11
3	TOM	TX	12
4	BOB	TX	13

Design time filter applied (AGE < 13) of left/central table

STATE

STATE	SNAME
MI	MICHIGAN
AL	ALABAMA

Design time filter applied to (STATE = MI) on right table

RESULT

C_ID	CNAME	STATE
1	WERNER	MICHIGAN
2	MARK	MICHIGAN
3	TOM	NULL

Filters are applied to both tables and then afterwards the join is executed.
Due to the left outer join TOM will be included in the result set even though he resides in TX.

Figure 29: Left Outer Joins and Design Time Filters

The figure, Left Outer Joins and Design Time Filters, shows the behavior of left outer joins with design time filters.

Left Outer Join in a CUBE Calculation View

```
Select
    o.ORDER_ID, o.C_ID, sub1.CUSTOMER, sub1.AGE,
    sub1.STATE, sub1.STATETXT, o.AMOUNT
   FROM "SalesOrder" o
   Left Outer Join
       (Select
           c.C_ID, c.CNAME As CUSTOMER,
           c.AGE, c.STATE, SNAME as STATETXT
           FROM "Customer" c
           Left Outer Join "State" s
           On c.STATE = s.STATE) as sub1
       On o.C_ID = sub1.C_ID
ORDER BY o.ORDER_ID
```

	ORDER_ID	C_ID	CUSTOMER	AGE	STATE	STATETXT	AMOUNT
1	1	1	WERNER	10	MI	MICHIGAN	100
2	2	1	WERNER	10	MI	MICHIGAN	100
3	3	2	MARK	11	MI	MICHIGAN	100
4	4	4	BOB	13	TX	?	100
5	8	77	?	?	?	?	100

Customer *TOM* is not returned due to no corresponding sale item record in sales order table.

AMOUNT	ORDER_ID	C_ID
100	1	1
100	2	1
100	3	2
100	4	4
100	8	77

C_ID	CNAME	AGE	STATE
1	WERNER	10	MI
2	MARK	11	MI
~~3~~	~~TOM~~	~~12~~	~~TX~~
4	BOB	13	TX

STATE	SNAME
~~AL~~	~~ALABAMA~~
MI	MICHIGAN

Figure 30: Left Outer Joins in a CUBE Calculation View

The figure, Left Outer Joins in a CUBE Calculation View, shows the use of left outer joins in a CUBE Calculation View.

Compared with the Inner join, all the Sales order data (including those with no corresponding Customer information) is retrieved, but still an analysis of Sales by Customer or State will give irrelevant data.

Right Outer Join

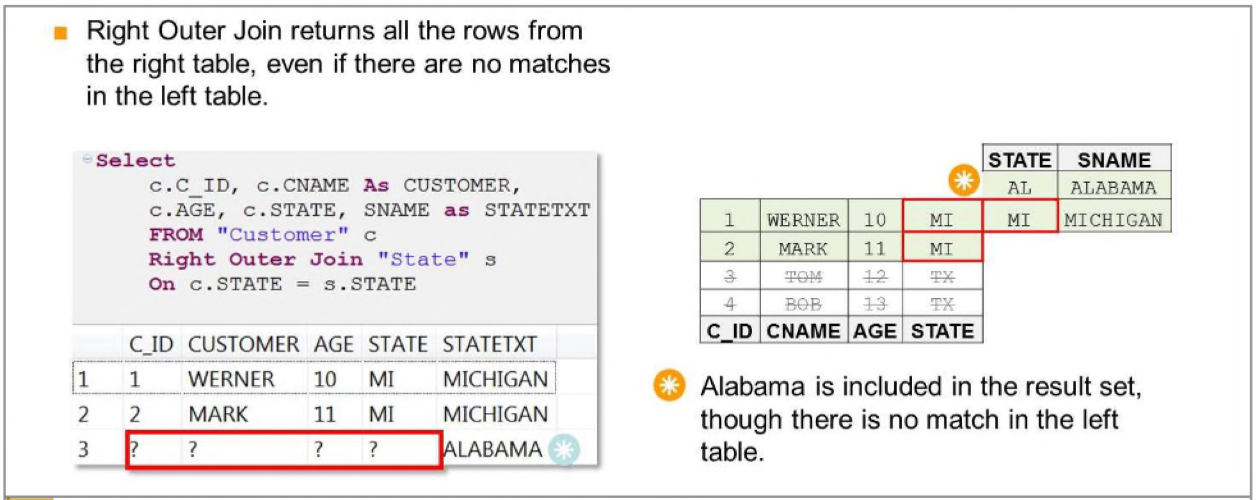


Figure 31: Right Outer Join in a DIMENSION Calculation View

A Right Outer Join returns all the rows from the right table, even if there are no matches in the left table.

Right Outer Join in a CUBE Calculation View

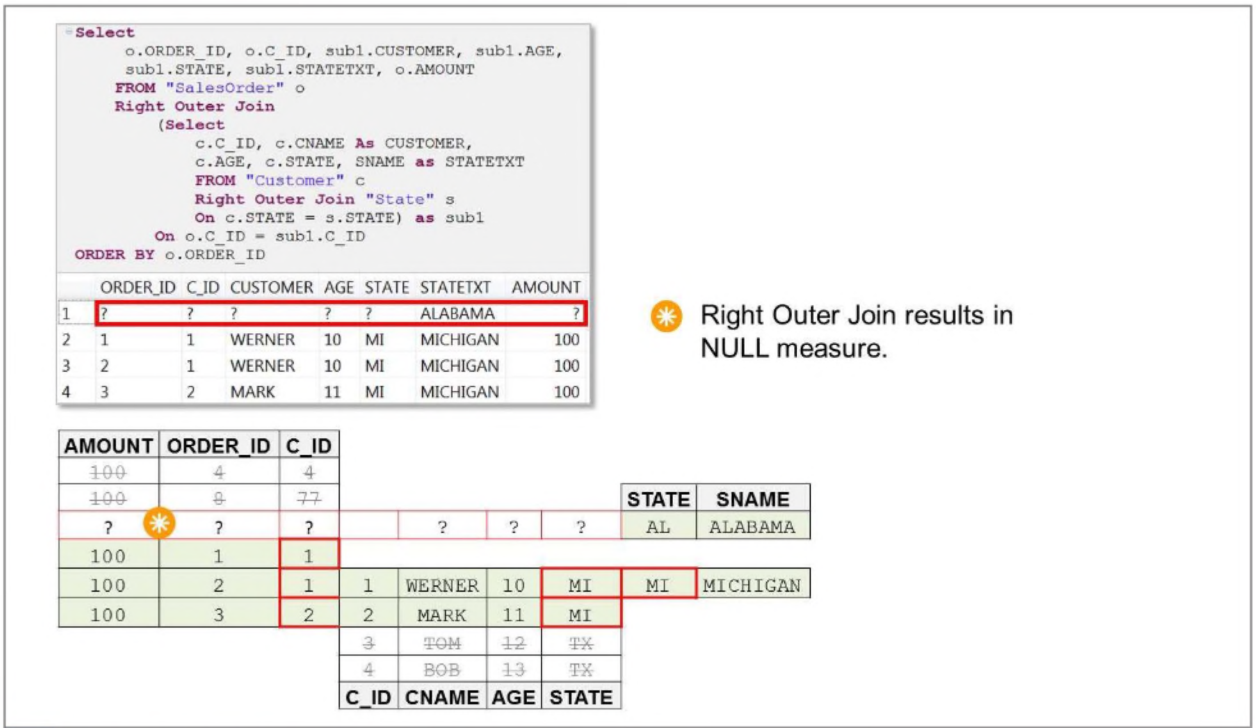


Figure 32: Right Outer Join in a CUBE Calculation View

Full Outer Join

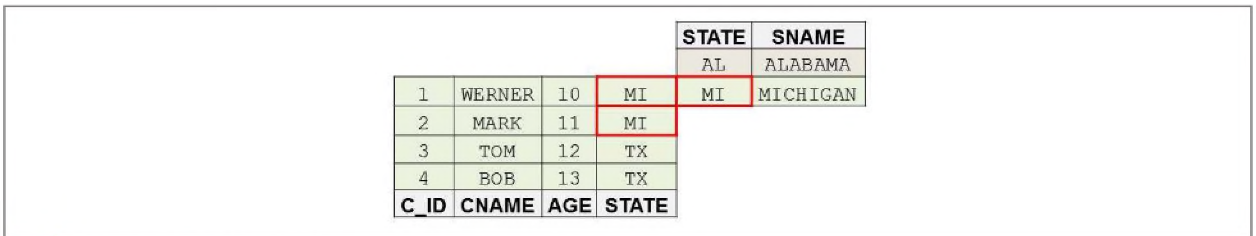


Figure 33: Full Outer Join

A full Outer Join (supported from SAP HANA service pack stacks (SPS) 11 onwards), combines the behaviors of the LEFT and RIGHT OUTER joins.

The result set is composed of:

- Rows from both tables that are matching on joined columns.
- Rows from the left table with no match in the right table.
- Rows from the right table with no match in the left table.

> Caution:
> Full Outer join is supported by Calculation Views only, in standard Join and Star Join nodes.
>
> However, in a Star Join node, a Full Outer join can be defined only on one DIMENSION calculation view, and this view must appear last in the star join node.

Referential Join

The referential join is semantically an inner join that assumes that referential integrity is given, meaning that the left table always has a matching entry in the right table.

It is an optimized or faster inner join where the right table is generally not checked if no field from the right table is requested.

> Note:
> From SPS11 onwards, the Referential Join is supported in any type of joins in Calculation Views. Before this version, you could only use it in a Star Join Node.

Relies on Referential Integrity	Each entry in the left table MUST have a corresponding entry in the right table
Optimized for performance	Join is only performed if at least one field from the right table is requested.
Like an Inner Join when join is executed	When field from both tables are requested an inner Join is performed.

Figure 34: Referential Join

In Calculation Views, referential joins are executed in the following way:

- If at least one field is selected from the right table, it will behave as an inner join.
- If no field from the right table is selected, the execution of the referential join depends on the cardinality:
  - If the cardinality is 1..1 or n..1, the join is not executed.

This corresponds to the most frequent situation, and in particular when you join a (left) fact table or view with a (right) view or table that contains only dimensions.

This is where the optimization occurs.

- In the rare case where you use a 1..n cardinality, the join is executed as an inner join.

  Indeed, this is a requirement to get the correct number of rows in the output, which depends on the number of matching rows in the right table or view.

From a performance perspective, the left outer join is almost equally fast as referential join, while the inner join is slower due to the join always being executed.

- Referential joins must be used with caution since they assume that referential integrity is ensured at any time.

> Note:
> If you consider a join between a fact table and its related attributes, keep in mind that facts without corresponding attributes violate referential integrity. While attributes without facts (for example, a customer without any order) do not.

Using referential joins in a context where referential integrity is not ensured might lead to different results depending on whether you select columns from the right table or not.

Referential Joins in a CUBE Calculation View

SELECT C_ID, CNAME, STATE, AGE, SNAME

C_ID	CNAME	STATE	AGE	SNAME
1	WERNER	MI	10	MICHIGAN
2	MARK	MI	11	MICHIGAN

*** Customer 77, TOM and BOB are not returned since C_ID is a Joined key field resulting in an Inner Join. BOB has no corresponding description for TX and TOM has no corresponding facts.

SELECT C_ID, CNAME, AGE

C_ID	CNAME	AGE
1	WERNER	10
2	MARK	11
4	BOB	13

*** TOM is not returned because there are no corresponding facts in the sales table.

SELECT SUM(AMOUNT)

AMOUNT
500

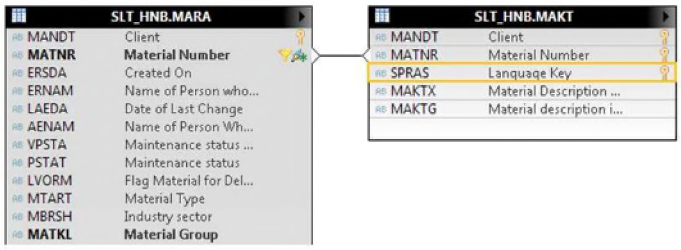
*** The Amount includes all facts including Customer 77 and BOBs order even though master records do not exist: when only non-key fields are selected from the left table, all joins to other tables will be omitted.

Figure 35: Referential Joins in a CUBE Calculation View

The figure, Referential Joins in a CUBE Calculation View, shows an example of referential joins in CUBE Calculation View.

Text Join

- Text Join are used to join a text table to a master data table.
- Text Joins acts as a Left Outer join and can be used with SAP tables where the language column (SPRAS) is present.
- For each attribute, it is possible to define a description mapping that will be specific to the end-user's language.



Property	Value
Left Column	
Right Column	
Cardinality	1..1
Join Type	Text Join
Left Table	SLT_HNB:MARA
Right Table	SLT_HNB:MAKT
Language Column	SPRAS

Property	Value
Attribute Name	MATNR
Description	Material Number
Key Mapping	SLT_HNB.MARA.MATNR
Description Mapping	SLT_HNB.MAKT.MAKTX

Figure 36: Text Joins

The text join enables SAP HANA to handle the translation of attribute labels in a way that corresponds to the way translation texts are stored in the master data or SAP Systems, such as SAP ERP.

Text Join Example

- Text join is used when a translation for a dimension is available
- Designed for ERP tables (and typically SPRAS column)
- User language is used as a filter at runtime to find the right translation for that attribute

PRODUCT table

SPRAS	ID	DESC
E	1	Car
D	1	Auto
E	2	Motorbike
D	2	Motorrad

Text join applied on **ORDER.PR_ID = PRODUCT.ID** using **SPRAS** as the language column

ORDER table

ORDER_ID	PR_ID	VALUE
00215753	1	1000
00237469	2	2000

Figure 37: Text Join Example

The figure, Text Join Example, is a simplified example of text join. Depending on the session language of the end-user, the *DESC* column displays the product description in English or in German.

Temporal Join

It is possible to add a temporal condition to a join, in order to find matching records from two tables, based on a date. The records are matched only if a date column of one table is within a time interval defined by two columns of the other table.

This is useful to manage time-dependent attributes.

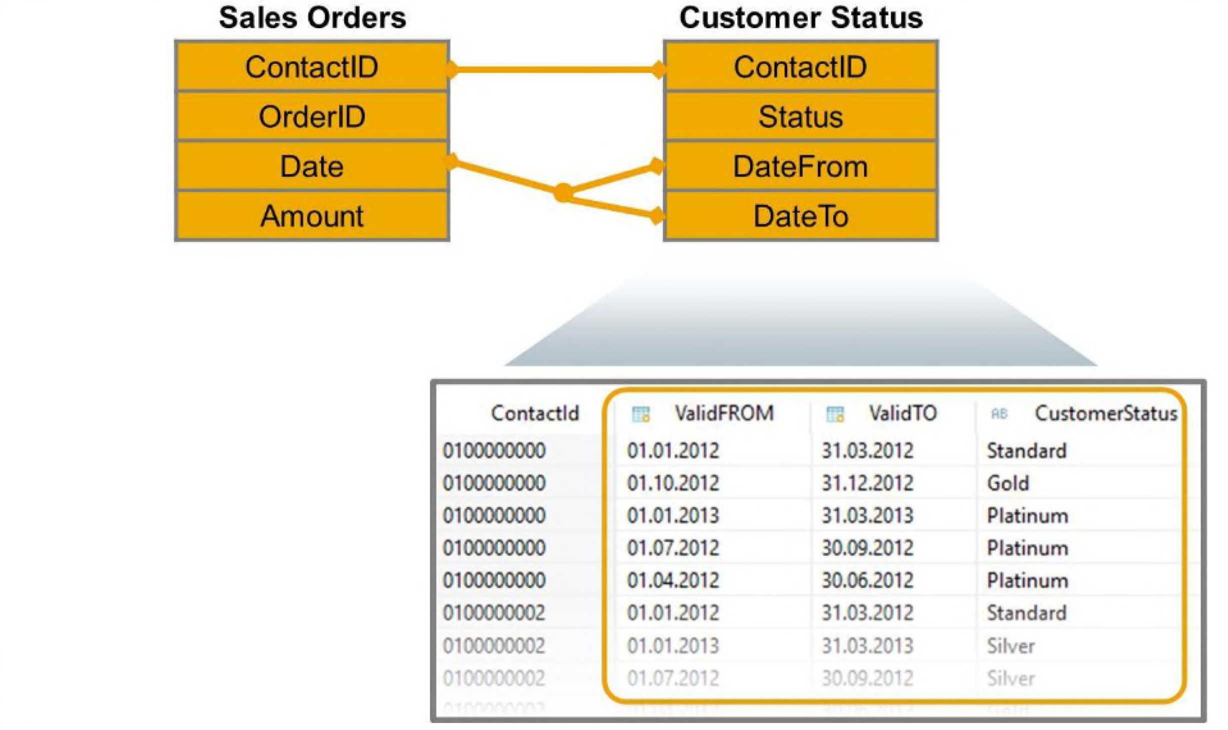


Figure 38: Temporal Join Example

In this example, the status of the customers can change over time, and this information is captured in a dedicated table (Customer Status). If you need to analyze the sales orders and include the status of each customer when he issued the order, you create an inner join on the *ContactID* column and add a temporal condition as follows:

- Temporal column: Date (Sales Orders)
- From Column: DateFrom (Customer Status)
- To Column: DateTo (Customer Status)
- Temporal Condition: Include Both

Temporal conditions can be added to the following joins types and view types:

- A Join or Star Join node of a Calculation View. The join type must be defined as *Inner*.
- The Star Join node of Analytic Views (deprecated view type). The join type must be defined as *Inner* or *Referential*.

> Note:
> Temporal conditions can be defined on columns of the following data types: *timestamp*, *date*, and *integer*.

Star join

The Star Join in Calculation Views of type *CUBE with Star Join* is a Node Type, rather than a Join type.

It is used to structure data in a star schema. The fact table (data source) of a star join can be any type of input node. However, only Calculation Views of Data Category *Dimension* are allowed as input nodes for dimensions.

The type of joins between the fact and dimension tables within the star schema can be defined in the star join node. The available joins are as follows:

- Referential join
- Inner join
- Left outer join
- Right outer join
- Full outer join (new join type in SAP HANA SPS11, with some specific restrictions when used in a Star Join)
- Text join

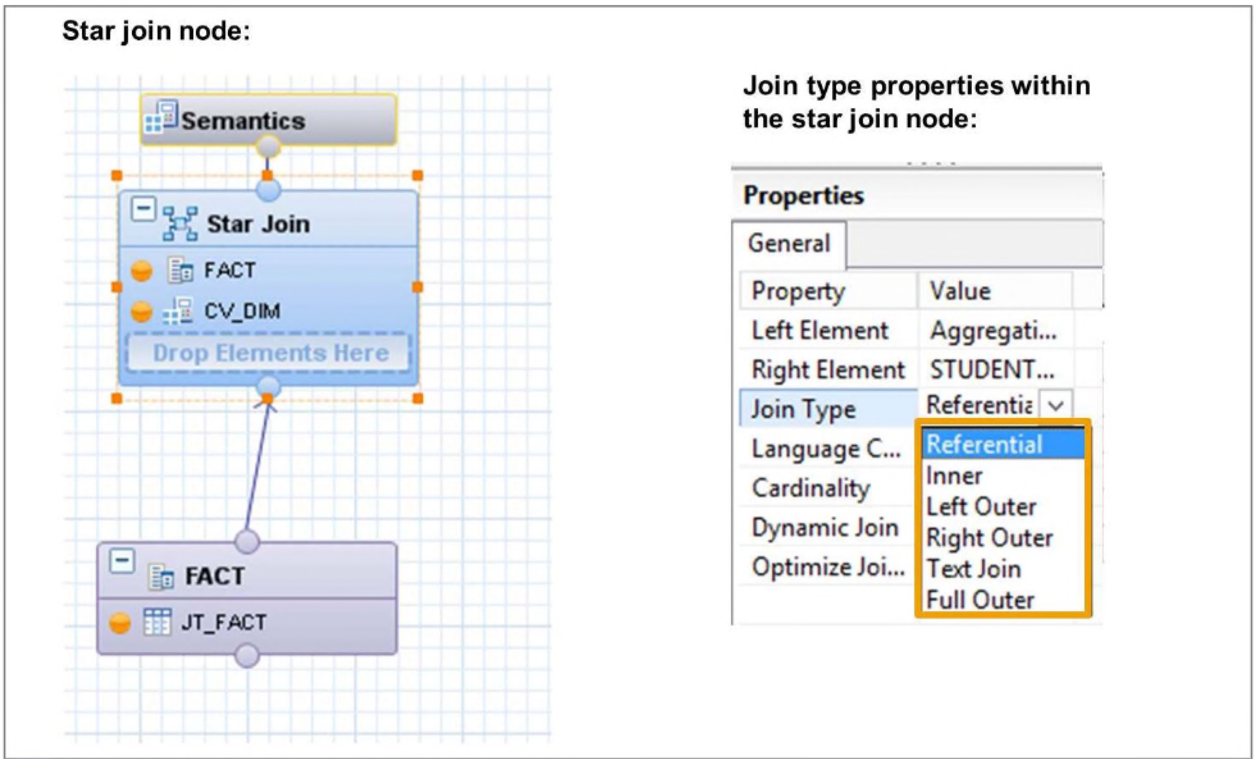


Figure 39: Star Join

Spatial Join

Introduced in SAP HANA SPS09, the Spatial Join type enables the modeler to use the native spatial capabilities of SAP HANA in Graphical modeling tools.

Note:
Up to version 1.0 SPS08, spatial joins could only be performed by SQL script, but not in graphical information views.

Spatial data modeling is covered in a dedicated lesson.

Join Cardinality

The cardinality of a join defines how the data from two tables joined together are related, in terms of matching rows.

For example, if you join the Sale Order table (left table) with the Customer table (right table), you can define an *n..1* cardinality. This cardinality means that several sales orders can be related to the same customer, but the contrary is not possible (you cannot have a sales orders that relates to several customers).

Caution:
It is recommended to specify the cardinality only when you are sure of the content of the tables. If not, just leave the cardinality blank.

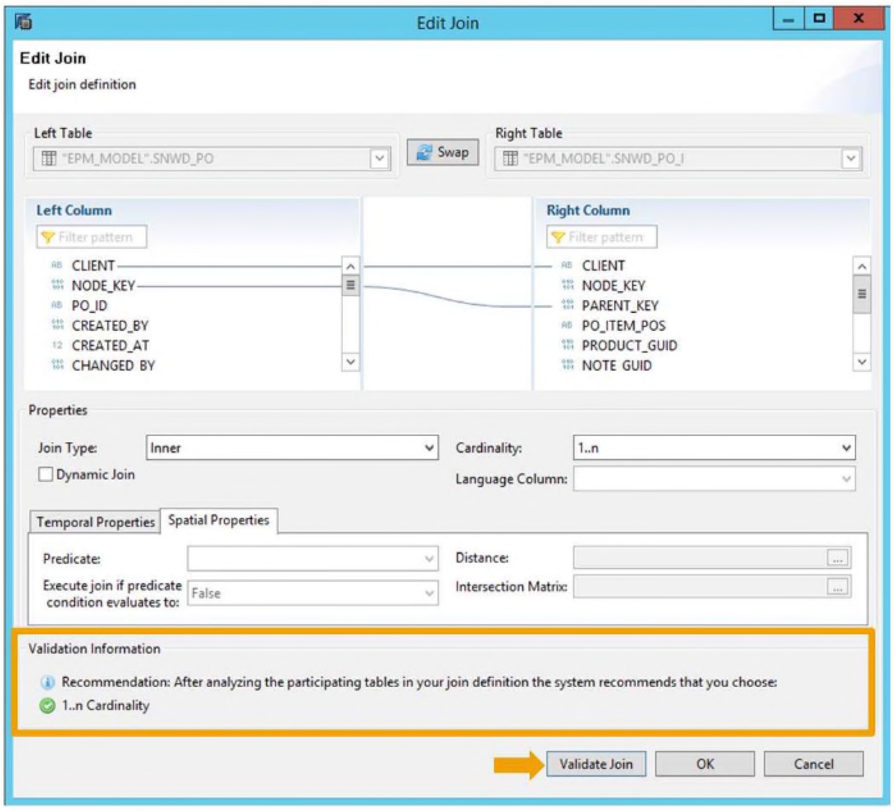


Figure 40: Validating a Join

From version SPS09 onwards, a new feature in SAP HANA studio suggests the recommended Cardinality, based on an analysis of the tables that are joined together.

> **Caution:**
> This analysis of joined tables is performed at the moment you define the join. If the content of the table evolves after that, the cardinality you have defined might become wrong.
>
> The follow is an example:
>
> You are validating the join between the *Sales Order* and *Customer* tables, but your data contain only one sales order per customer. In this case, the join validation might suggest a *1..1* cardinality, which does not correspond to the expected scenario in real life.

Combining Several Data Sets: Union vs. Join

When you need to combine measures originating from more than one table or Calculation View, you might want to create a join between these tables. However, under most circumstances, you must avoid using a join to address this requirement, because it is very costly from a computing standpoint.

It is more beneficial to use a union node, which provides much better performance.

> **Note:**
> Technically, a union is not a join type.

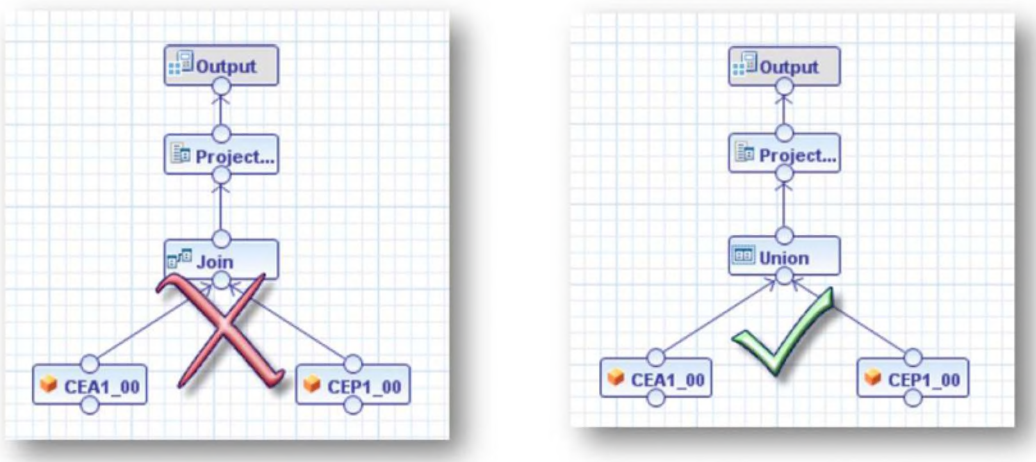


Figure 41: Do NOT Combine Data Sets with Joins in Calculation Views

LESSON SUMMARY

You should now be able to:

- Connect Tables
- Determine the type of join to use when connecting tables

Unit 1
Lesson 3

Creating Dimension Calculation Views

LESSON OVERVIEW

This lesson covers the creation of Dimension Calculation Views, which are used to organize the master data available in the underlying database tables. These views are, in turn, joined to fact table in Cube Calculation Views to analyze the facts in a way that suits your reporting needs.

Dimension Calculation Views can be shared by several Cube calculation views, which makes the maintenance of the virtual data model much easier.

Business Example

You want to create a view on your customer information (for example, ID, name, country, city, address, category, and so on) based on different tables.

In SAP HANA, you generally retrieve this information with Calculation View of type DIMENSION, which enables you to structure the master data and adapt it to your reporting requirements, for example:

- Join several tables to define a dimension that contains all the attributes you need
- Select only some of the columns from the source tables, as some data could be unnecessary
- Modify the name or labels of the columns
- Semantically associate the name or code and the description of objects (for example, product code and product description)
- Define calculated attributes
- Define hierarchies based on the master data

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create Dimension Calculation views
- Define calculated attributes
- Create time-based dimension views
- Use base table aliases
- Define label columns and hide attributes in a Dimension Calculation View

Dimension Calculation Views

Dimension Calculation Views are used to give context. This context is provided by text tables which give meaning to data.

For example, if a fact table or CUBE calculation view in a car sales database only contains a numeric ID for each dealer, you can use a Dimension Calculation View to provide information about each dealer. Using this method, you could then display the names and addresses of the car dealers, and give context to the data.

Dimension Calculation Views are used to select a subset of master data columns and rows from tables. They are not meant to handle measures or aggregates.

Dimension Calculation View does not need to be based on a single table. On the contrary, you can use them to join master data tables to each other. For example, to join Products to Product Categories.

Shared Dimension Calculation Views

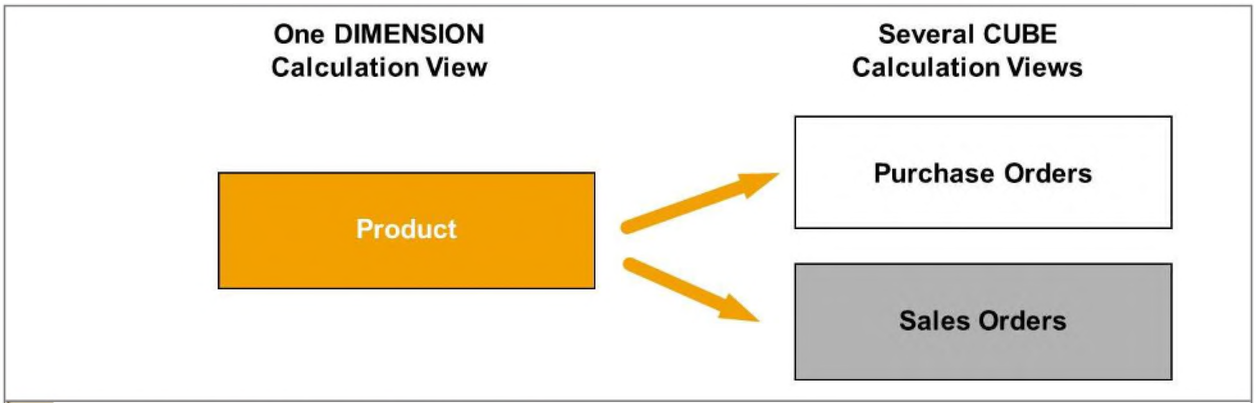


Figure 42: Shared Dimension Calculation Views

Like many other types of information views, DIMENSION Calculation Views are reusable objects and can be shared between several CUBE calculation views.

For example, the product attribute view can be used both in a purchase order CUBE Calculation view and in a sales order CUBE Calculation view.

Supported Characters for Views Names and View Objects

SAP HANA supports all the UNICODE characters in object names, that is views, columns, input parameters, and so on.

However, a number of characters are forbidden in the object names. These characters are:

`\ / : * ?" < > | . ; ' $ % , ! # +` and the **space** character.

Supported Types of Nodes

Dimension Calculation Views support all types of nodes (Projection, Join, Union, Aggregation, and Ranking). This allows a very flexible design of DIMENSION calculation view. The most commonly used nodes are Join nodes (to join Master Data tables) and Projection nodes (to filter data, select specific columns from the Master Data tables, and create calculated attribute columns).

Only the Star Join node is not supported in this type of view, because it is specific to CUBE with Star Join calculation views (and Analytic Views, which are now deprecated).

Calculated Attributes

It is possible to create additional calculated columns in an Dimension Calculation View.

For example, you have two columns containing the first and last name of the customer, but you would like to have all this information (first and last name) in a single column. You can do this by creating a calculated column based on string manipulations.

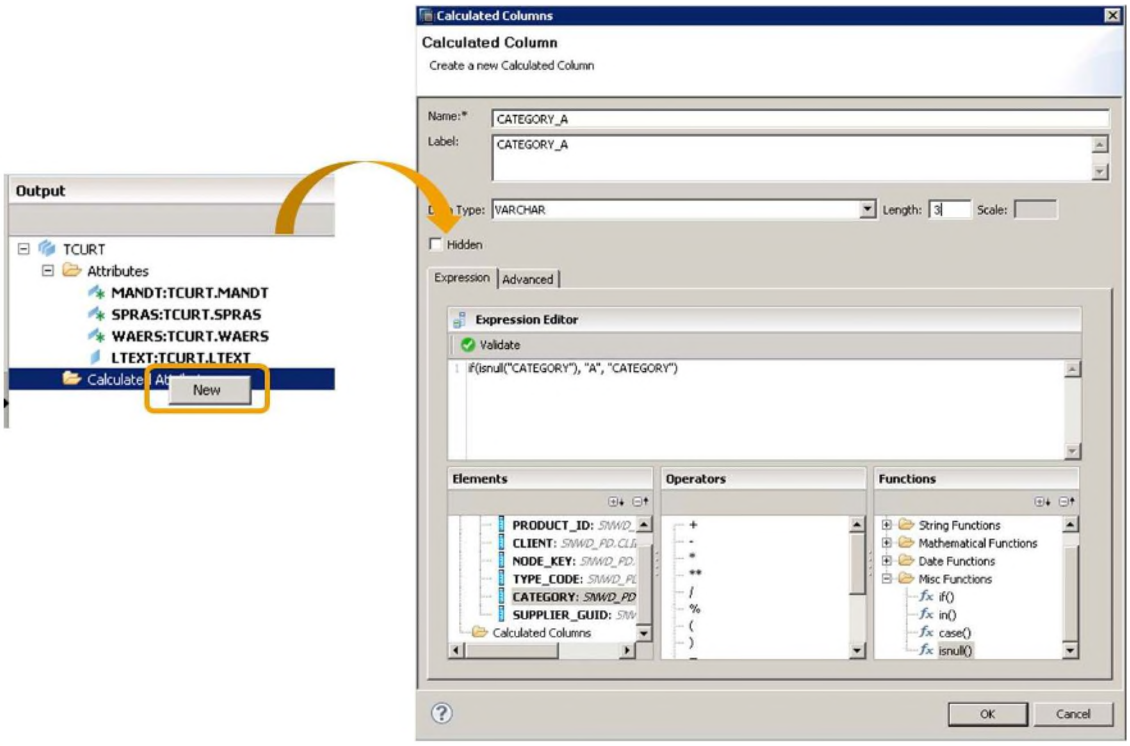


Figure 43: Calculated Columns

- The calculation can be an arithmetic or just a character string manipulation.
- Calculated columns also support non-measure attributes as part of the calculation.
- It is possible to nest calculated columns, so that one calculated column in turn is based on other calculated columns.

Time-Based Dimension Calculation Views

Apart from the *Standard* type for Calculation Views, the *Time* type, is used to create Dimension Calculation Views that apply exclusively to time data.

The tables containing measures generally have date or date-time columns to clearly define when an event occurred, or what period of the year, such as a month, or a quarter, a measure relates to. The purpose of these Time-Based Dimension Views is to ease the manipulation of measures across time.

In order to report in a simple and flexible way, you can use time-based Dimension Views to build a time hierarchy that correspond to your needs, convert a timestamp into simpler attributes, such as year, month, or day.

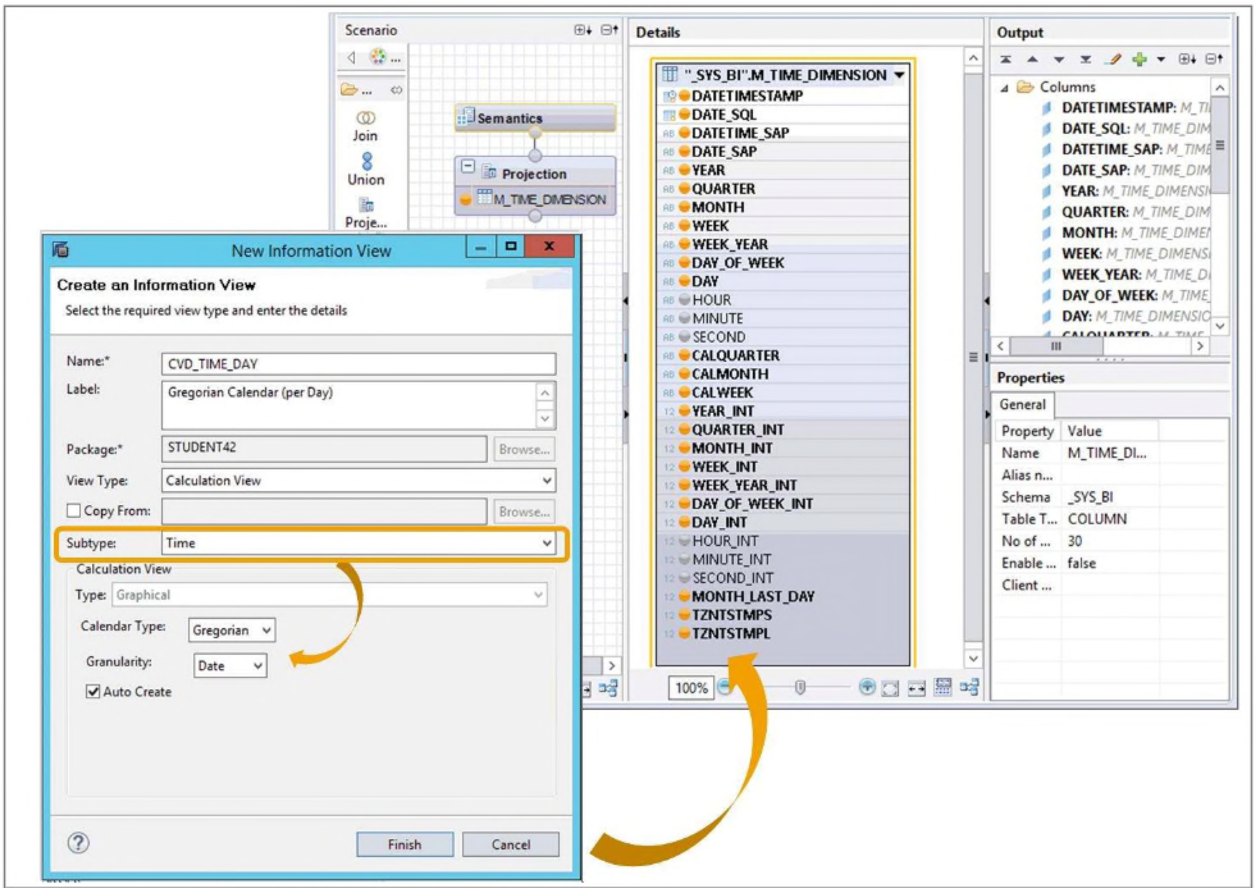


Figure 44: Time-Based Dimension Calculation Views

> Note:
> Even if it does not appear explicitly in the user interface, a Calculation View of subtype *Time* is automatically created as a *DIMENSION* Calculation View. You can check that in the *Properties* tab of the view.

The following are different Calendar types available in Time Calculation Views:

- Gregorian

  The Gregorian calendar is made up of years, months, and days. You can adjust the level of granularity, down to hours, minutes or seconds.

- Fiscal

  The fiscal calendar is organized into fiscal years and fiscal periods. Several fiscal year variants can be defined depending on your reporting needs.

> Note:
> The fiscal calendar is especially useful to display data in your information models according to the Fiscal Calendar tables available in your SAP ERP System.

To Create a Time Dimension Calculation View

The time data has been generated in your SAP HANA System.

> Note:
> This is generally done by a system administrator, and it populates some *M_TIME_DIMENSION_** tables (Gregorian calendar) or the *M_FISCAL_CALENDAR* table (Fiscal calendar), located in the *_SYS_BI* schema.

1. Create a new Calculation View of subtype *Time*.
2. Choose a calendar type (Gregorian or Fiscal).
3. Define the granularity level (Gregorian) or the variant (Fiscal).
4. To get the relevant table from the *_SYS_BI* schema automatically included in the calculation view scenario, select the *Auto Create* checkbox.
5. Choose Finish.
6. Add the columns that fit your reporting needs to the output.

   Make sure you include a relevant column that will be used as a key to join the time Calculation View to the tables or views that contain your measures.
7. If needed, organize time data into level hierarchies.
8. Save and activate your calculation view.

Base Table Alias

- When adding multiple instances of the same table to a view an alias will be proposed.
- It is possible to modify the alias name in the table properties.

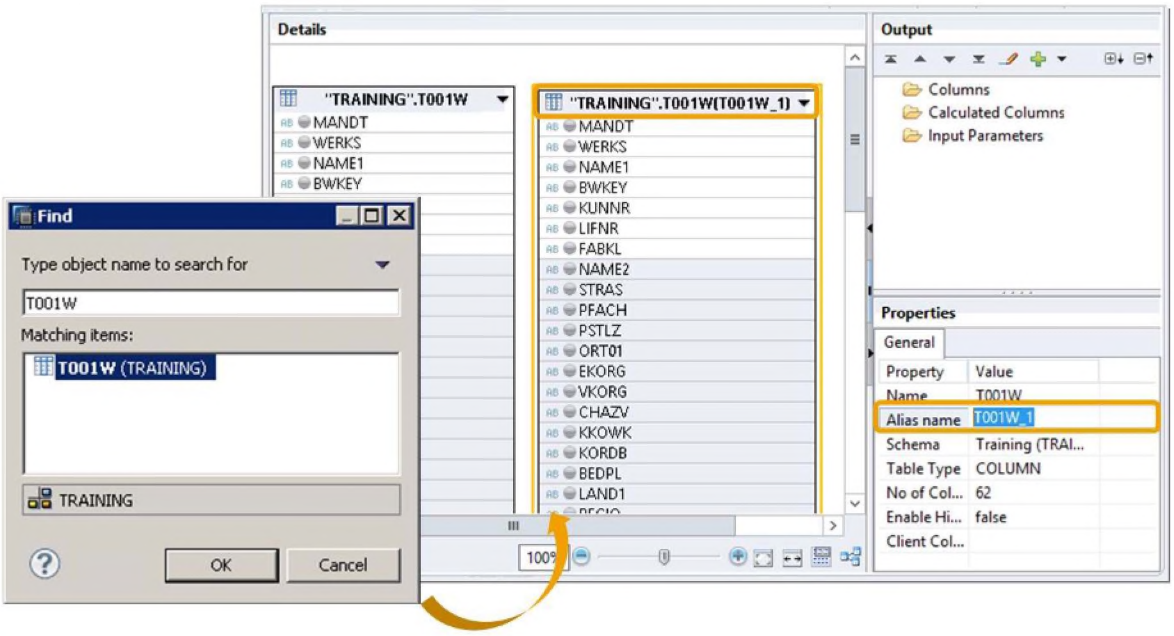


Figure 45: Adding Multiple Base Tables Using Aliases

In some cases, you need to use the same base table more than once in the same calculation view. In this case, you can define a table alias for any additional instance of the same source table.

> Note:
> SAP HANA studio automatically suggests an alias, but you can choose your own.

Hidden Columns

- To hide a column, select the *Hidden* checkbox of the column in the *Semantics* node.
- The column will not be exposed to reporting tools

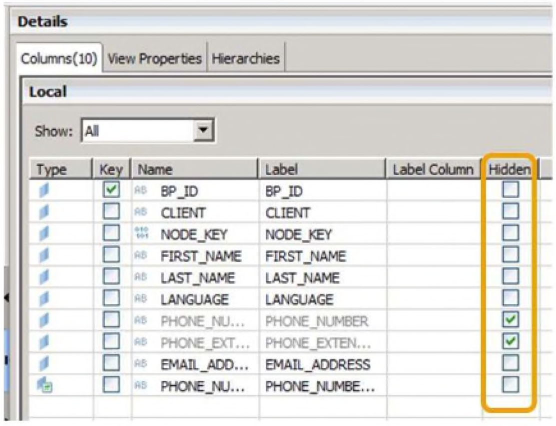


Figure 46: Hidden Columns

You can choose to hide the attributes that are not required for client consumption. This typically occurs in the following scenarios:

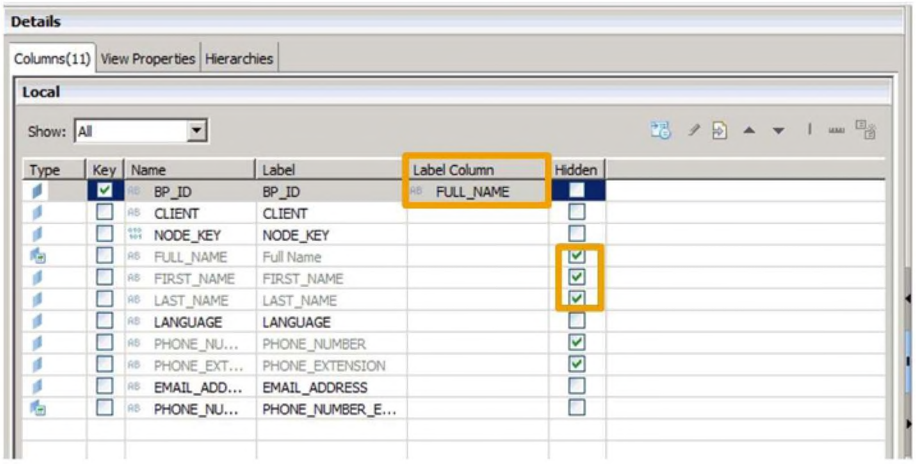
- When an attribute column is calculated using a complex computation of source columns or intermediary calculated columns, you can hide any column that you do not want to expose to the end user.
- When a column that contains a description has been assigned to the corresponding code, name or ID column as a *Label Column*, you can hide this column. This action is particularly useful in scenarios where the reporting tools are able to interpret this code, name or ID to description assignment.

  For example, SAP BusinessObjects Analysis and SAP Design Studio are able to expose *Label Columns*. The code, name, or ID, and the label, are generally referred to as *Key* and *Text*, respectively.

> Note:
> Only a column that is added to the output of the top node of a calculation view can be defined as a *Label Column*.

Label Columns and Hidden Attributes

- Label columns can refer to hidden attributes
- You define label columns and hidden attributes in the semantics



When reporting on an information model containg a label column, with a tool that supports Label Columns:

- Label is displayed as "Text".
- The base column is displayed as "Key"

Figure 47: Label Columns and Hidden Attributes

General Properties of Views

For each view, you can define a number of Properties in the *View Properties* tab of the *Details* pane. Depending on the type of view (Graphical or Script-Based Calculation View), the list of available properties may vary.

The following table gives a list of these properties and a definition. You will find more information on some of them in this course later on, and you can also consult the SAP HANA Documentation, in particular the *SAP HANA Modeling Guide*.

Table 3: General Properties of Views

Property	Description
Data Category	For Calculation Views, determines whether the view supports multi-dimensional reporting.
Type	Standard, Time or Derived.*
Base Attribute View*	For a derived Attribute View, identifies the source Attribute View from which it is derived.
Default Client	Defines how to filter data by SAP client (MANDT).
Run With	Defines how to apply security when executing a script-based calculation view.
Default Schema	Defines which schema must be used when the source schema of a catalog object is not specified.
Apply Privileges	Specifies the type of Analytic Privileges (Standard or Structured Query Language [SQL]) to apply when executing a view.
Enable History	The value of this property determines whether your calculation view supports time travel queries.

Property	Description
History Input Parameter	Specifies which Input Parameter must be used to specify the timestamp in time travel queries.
Column View Name	This field contains the full name of the column view in the _*SYS_BIC* schema. You can easily copy and paste this name to the code of a SQL statement.
Deprecate	Identifies views that are not recommended for reuse, though still supported in SAP HANA Modeler.
Cache	Defines whether the data retrieved by the view should be cached.
Cache Invalidation Period	If the view data are cached, determines whether the cache must be deleted on a daily or hourly basis, or after each transaction that modifies any of the underlying tables.
Translate	Determines whether the view labels can be translated into one or several additional languages.
Execute in	Determines whether the model must be executed by the SQL engine or column engine.
Pruning Configuration Table	Identifies which table contains the settings to prune Union nodes.

Note:
(*) The Derived Attribute View type is specific to Attribute Views. There is no equivalent for DIMENSION Calculation Views.

Unit 1
Exercise 2

Create a Dimension Calculation View

Exercise Objectives

After completing this exercise, you will be able to:

- Create a Dimension Calculation View
- Create a Calculated Attribute in a Dimension Calculation View

Business Example

You are at a customer site where the Enterprise Procurement Model (EPM) is used. Information about business partners, products, sales orders, and purchase orders is located in several tables, and you need to structure this data in a report.

The first step is to create two Dimension Calculation Views for the *Business Partner* dimension. This view will be used later on to create the Star Join Calculation Views for Purchase Orders.

The view will join several master data tables related to the business partners, including the contact persons and their email.

> Note:
> In this exercise, when the values include ##, replace the characters with the number that your instructor assigned you.

1. In case you are not already in the *SAP HANA Modeler* perspective, open it.

   In order to have more space on your screen, you can close the *Quick View*, because it is not required.

2. In your *STUDENT##* package, create a new Calculation View with the following properties:

Field	Value
Name	**CVD_BP_##** (## is your student number)
Description	**Business Partners**
Subtype	*Standard*
Calculation View Type	*Graphical*
Category	*DIMENSION*

3. Add a Join Node to the *Scenario* pane, and add the following data sources to the new node:

- Table *SNWD_BP (schema EPM_MODEL)* (Business Partners master data)
- Table *SNWD_CONTACT (schema EPM_MODEL)* (Contact person details)

Note:
The new Join node is automatically called *Join_1*

4. In the *Details* pane, join the data sources of node *Join_1* as follows, for each join use a *Referential* join type and a *1..1* cardinality:

   The table *SNWD_BP* is considered as the left table, and the table *SNWD_CONTACT* as the right table.

   - Join the field *CLIENT* of the left table to the field *CLIENT* of the right table.
   - Join the field *NODE_KEY* of the left table to the field *PARTNER_GUID* of the right table.

Hint:
To check the join type and cardinality, in the *Edit Join* window, you can choose *Validate Join*.

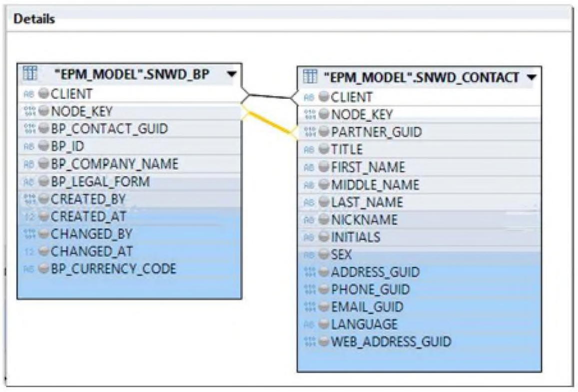


Figure 48: Creating Joins

5. In the *Details* pane for node *Join_1*, add the following columns to the output.

Source table	Columns
SNWD_BP	• BP_ID • BP_COMPANY_NAME • CLIENT • NODE_KEY
SNWD_CONTACT	• FIRST_NAME • LAST_NAME • LANGUAGE

6. Add another Join Node *Join_2* to the *Scenario* pane, between the nodes *Join_1* and *Projection*, and add the two following data sources to this new node:

   - Node *Join_1*
   - Table *EPM_MODEL.SNWD_BP_EM* (e-mail address)

> Hint:
> To link a node to another one, just draw a connector from the dot on top of the source node to the dot at the bottom of the target node.

7. In the *Details* pane, join the data sources of node *Join_2* as follows, using for each join a *Referential* join type and a *1..1* cardinality:

   The source *Join_1* is considered as the left table, and the table *EPM_MODEL.SNWD_BP_EM* as the right table.

   - Join the field *CLIENT* of the left table to the field *CLIENT* of the right table.
   - Join the field *NODE_KEY* of the left table to the field *PARTNER_GUID* of the right table.

> Note:
> The join is defined on the same fields as the previous one.

8. In the *Details* pane for node *Join_2*, add the following columns to the output.

Source Table	Columns
Join_1	[All the columns]
SNWD_BP_EM	EMAIL_ADDRESS

> Hint:
> To add all the columns from *Join_1* in a single step, right-click the *Join_1* table.

9. Define the *Join_2* node as the data source for the *Projection* node, and add all the columns of the *Projection* node to the output.

10. Define the *BP_ID* and column as Key Attribute, and all the other columns as Attribute.

11. In the *Projection* node, create a Calculated Column *FULL_NAME* that combines the first and last name of the business partners.

    The new column must be defined as follows:

Field	Value
Name	FULL_NAME
Label	FULL_NAME

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

Field	Value
Data Type	VARCHAR
Length	100
Expression	"FIRST_NAME"+' '+"LAST_NAME"

12. Save and activate the Calculation View.

    Check the activation status after activation.

13. Preview the data of the *CVD_BP_##* Calculation View.

14. Close the *CVD_BP_##* Calculation View.

Unit 1
Solution 2

Create a Dimension Calculation View

Exercise Objectives

After completing this exercise, you will be able to:

- Create a Dimension Calculation View
- Create a Calculated Attribute in a Dimension Calculation View

Business Example

You are at a customer site where the Enterprise Procurement Model (EPM) is used. Information about business partners, products, sales orders, and purchase orders is located in several tables, and you need to structure this data in a report.

The first step is to create two Dimension Calculation Views for the *Business Partner* dimension. This view will be used later on to create the Star Join Calculation Views for Purchase Orders.

The view will join several master data tables related to the business partners, including the contact persons and their email.

> Note:
> In this exercise, when the values include ##, replace the characters with the number that your instructor assigned you.

1. In case you are not already in the *SAP HANA Modeler* perspective, open it.

   In order to have more space on your screen, you can close the *Quick View*, because it is not required.

   a) Choose *Window* → *Open Perspective* → *Other* → *SAP HANA Modeler*.

   b) If needed, close the *Quick View*, on the right of the tab name, choose *Close*.

2. In your *STUDENT##* package, create a new Calculation View with the following properties:

Field	Value
Name	**CVD_BP_##** (## is your student number)
Description	**Business Partners**
Subtype	*Standard*
Calculation View Type	*Graphical*
Category	*DIMENSION*

a) In the *Content* folder, right-click your *STUDENT##* package and choose *New → Calculation View*.

b) Enter the Calculation View name and other properties as specified in the table.

c) Choose *Finish*.

The Calculation View graphical editor opens on the right of your screen.

3. Add a Join Node to the *Scenario* pane, and add the following data sources to the new node:

- Table *SNWD_BP (schema EPM_MODEL)* (Business Partners master data)
- Table *SNWD_CONTACT (schema EPM_MODEL)* (Contact person details)

> Note:
> The new Join node is automatically called *Join_1*

a) In the *Scenario* pane, add a join node below the top nodes by dragging a join node from the palette.

b) Hover over the new join node and click the + sign on the right of the *Join_1* node.

c) In the search field, enter **SNWD_**.

d) To select the tables, on your keyboard, press Ctrl and select the tables.

e) Choose *OK*.

4. In the *Details* pane, join the data sources of node *Join_1* as follows, for each join use a *Referential* join type and a *1..1* cardinality:

The table *SNWD_BP* is considered as the left table, and the table *SNWD_CONTACT* as the right table.

- Join the field *CLIENT* of the left table to the field *CLIENT* of the right table.
- Join the field *NODE_KEY* of the left table to the field *PARTNER_GUID* of the right table.

> Hint:
> To check the join type and cardinality, in the *Edit Join* window, you can choose *Validate Join*.

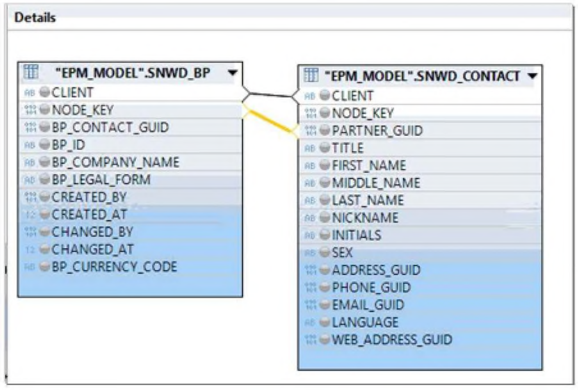


Figure 48: Creating Joins

a) In the *Details* pane, drag the *SNWD_BP.CLIENT* field to the *SNWD_CONTACT.CLIENT* field, and drag the *SNWD_BP.NODE_KEY* field to the *SNWD_CONTACT.PARTNER_GUID* field.

> Caution:
> For the second join, make sure you actually join the PARTNER_GUID of the right table (NOT the NODE_KEY).

b) To edit the join properties, double-click the new connector.

c) In the *Join Type* dropdown list, choose *Referential*.

d) In the *Cardinality* dropdown list, choose *1..1*.

e) Optionally, choose *Join Validation* to check the join properties.

f) Choose *OK*.

> Note:
> The *Join Type* and *Cardinality* properties are defined for each join, in this case, the pair of connections between the tables.

5. In the *Details* pane for node *Join_1*, add the following columns to the output.

Source table	Columns
SNWD_BP	• BP_ID • BP_COMPANY_NAME • CLIENT • NODE_KEY

Source table	Columns
SNWD_CONTACT	• FIRST_NAME • LAST_NAME • LANGUAGE

a) In the *EPM_MODEL.SNWD_BP* table, right-click the *BP_ID* field and choose *Add To Output*.

> Hint:
> Alternatively, you can double-click the field name or click toggle on the left of a table field.

b) Repeat the previous step for the other fields to be included in the output.

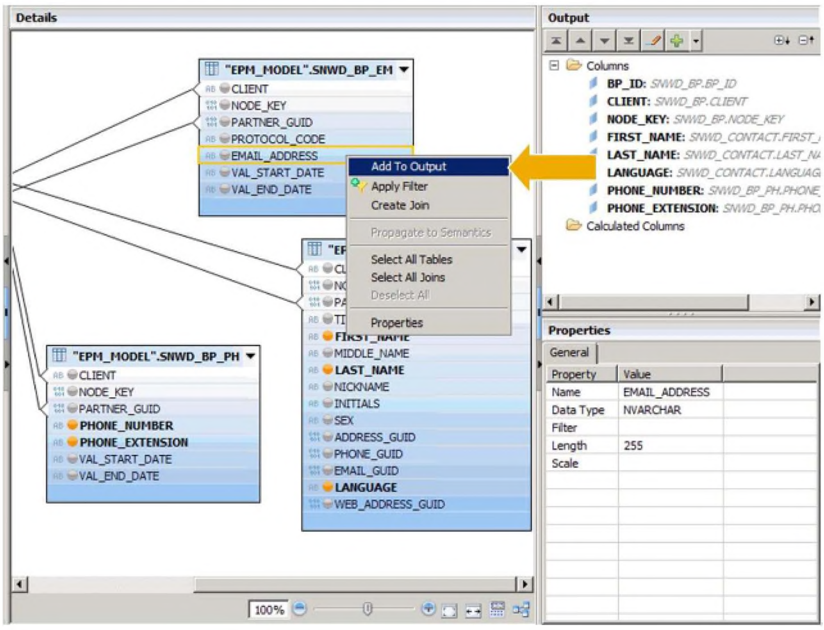


Figure 49: Adding Columns to Output

6. Add another Join Node *Join_2* to the *Scenario* pane, between the nodes *Join_1* and *Projection*, and add the two following data sources to this new node:

- Node *Join_1*
- Table *EPM_MODEL.SNWD_BP_EM* (e-mail address)

> Hint:
> To link a node to another one, just draw a connector from the dot on top of the source node to the dot at the bottom of the target node.

a) In the *Scenario* pane, draw a connector from the dot on top of the *Join_1* node to the dot at the bottom of the *Join_2* node.

b) Choose the + sign on the right of the *Join_2* node.

c) In the search field, enter **SNWD_**.

d) Select the *SNWD_BP_EM (EPM_MODEL)* table.

e) Choose *OK*.

7. In the *Details* pane, join the data sources of node *Join_2* as follows, using for each join a *Referential* join type and a *1..1* cardinality:

   The source *Join_1* is considered as the left table, and the table *EPM_MODEL.SNWD_BP_EM* as the right table.

   - Join the field *CLIENT* of the left table to the field *CLIENT* of the right table.
   - Join the field *NODE_KEY* of the left table to the field *PARTNER_GUID* of the right table.

> Note:
> The join is defined on the same fields as the previous one.

a) In the *Details* pane, drag the *Join_1.CLIENT* field to the *SNWD_BP_EM.CLIENT* field, and drag the *Join1.NODE_KEY* field to the *SNWD_BP_EM.PARTNER_GUID* field.

> Caution:
> For the second join, make sure you actually join the PARTNER_GUID of the right table (NOT the NODE_KEY).

b) To edit the join properties, double-click the new connector.

c) In the *Join Type* dropdown list, choose *Referential*.

d) In the *Cardinality* dropdown list, choose *1..1*.

e) Optionally, choose *Join Validation* to check the join properties.

f) Choose *OK*.

8. In the *Details* pane for node *Join_2*, add the following columns to the output.

Source Table	Columns
Join_1	[All the columns]
SNWD_BP_EM	EMAIL_ADDRESS

> Hint:
> To add all the columns from *Join_1* in a single step, right-click the *Join_1* table.

a) In the *Details* pane, right-click the title bar of the *Join_1* table and choose *Add All to Output*.

b) In the *EPM_MODEL.SNWD_BP_EM* table, right-click the *EMAIL_ADDRESS* field and choose *Add To Output*.

9. Define the *Join_2* node as the data source for the *Projection* node, and add all the columns of the *Projection* node to the output.

   a) In the *Scenario* pane, draw a connector from the dot on top of the *Join_2* node to the dot at the bottom of the *Projection* node.

   b) In the *Details* pane of the *Projection* node, right-click the title bar of the *Join_2* table and choose *Add All to Output*.

10. Define the *BP_ID* and column as Key Attribute, and all the other columns as Attribute.

    a) In the *Scenario* pane, choose the *Semantics* node.

    b) In the *Details* pane, select the *Key* checkbox of the *BP_ID* column.

11. In the *Projection* node, create a Calculated Column *FULL_NAME* that combines the first and last name of the business partners.

    The new column must be defined as follows:

Field	Value
Name	FULL_NAME
Label	FULL_NAME
Data Type	VARCHAR
Length	100
Expression	"FIRST_NAME"+' '+"LAST_NAME"

    a) Select the *Projection* node.

    b) In the *Output* pane, choose + *New Calculated Column...*.

    c) Enter the properties as shown in the table above.

    d) Choose *Validate Syntax*.

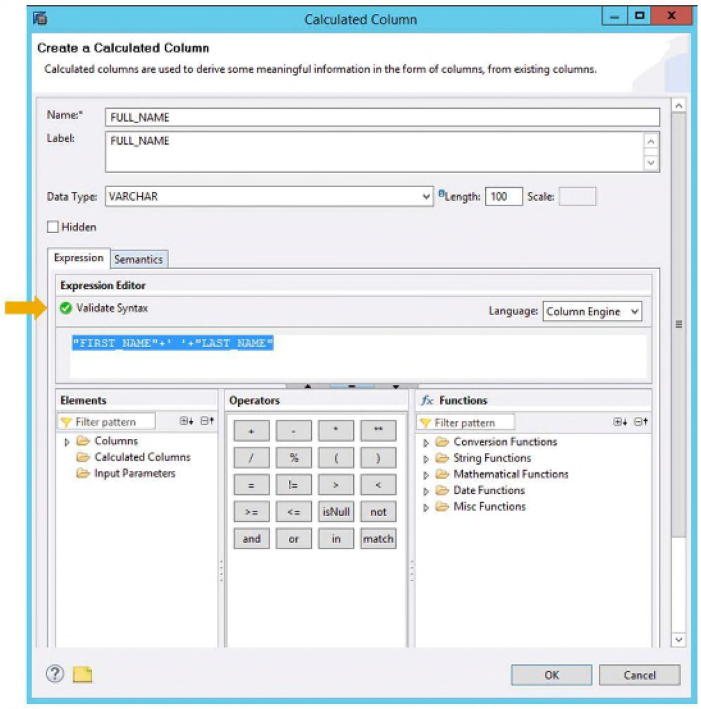


Figure 50: Creating a Calculated Column

e) Choose *OK*.
The new column is added to the Calculation View.

12. Save and activate the Calculation View.

Check the activation status after activation.

a) On the toolbar above the *Details* pane, choose *Save and Activate*.

b) In the *Job Log* view, on the *Current* tab, check the status of the last *Activation* job.

c) To display the activation *Job Details*, double-click the corresponding row in the *Job Log*.

d) If needed, review the *Detailed Report*.

e) Choose *OK*.

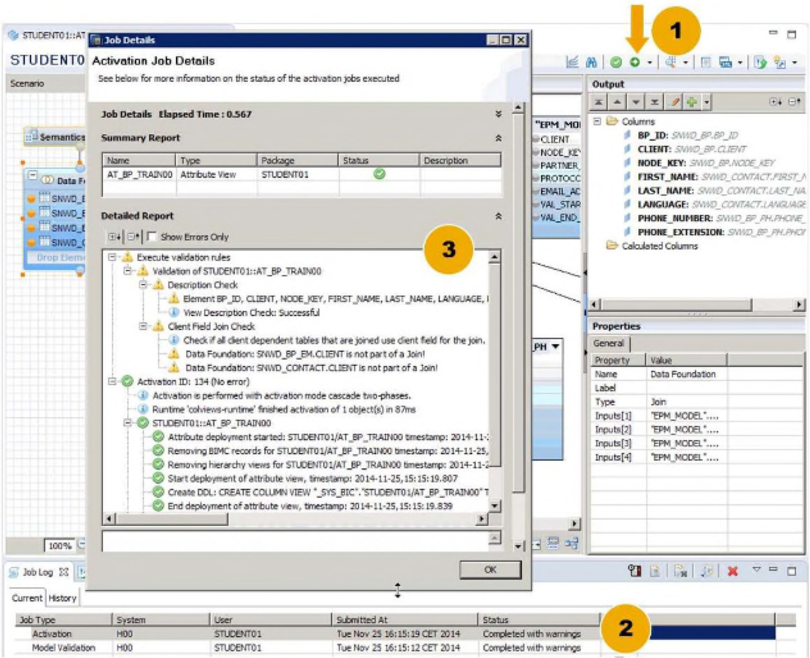


Figure 51: Saving and Activating the Calculation View

13. Preview the data of the *CVD_BP_##* Calculation View.

    a) On the toolbar above the *Details* pane, choose *Data Preview*, and choose *Open in Data Preview Editor*.

    b) Select the *Raw Data* tab.

    c) To close the data preview, choose *X (Close)*.

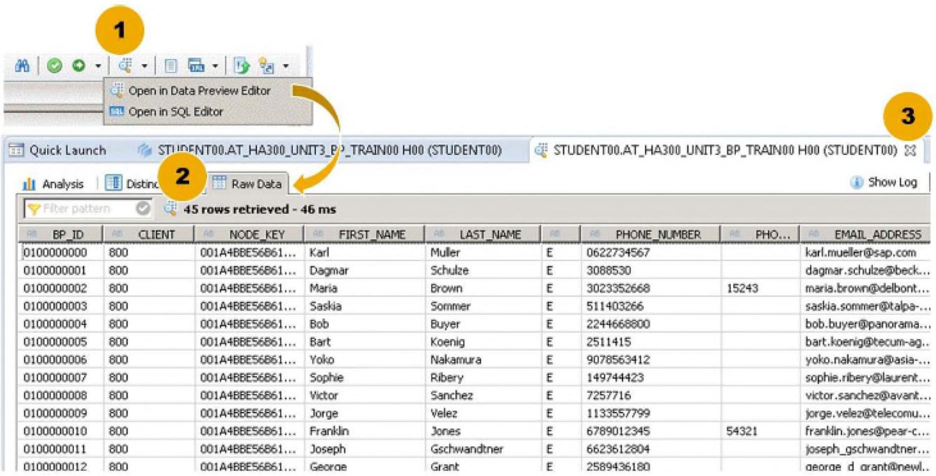


Figure 52: Data Preview

14. Close the *CVD_BP_##* Calculation View.

LESSON SUMMARY

You should now be able to:

- Create Dimension Calculation views
- Define calculated attributes
- Create time-based dimension views
- Use base table aliases
- Define label columns and hide attributes in a Dimension Calculation View

Unit 1
Lesson 4

Using Measures in Calculation Views

LESSON OVERVIEW

This lesson will give you an overview of how to handle measures in graphical Calculation Views, before you deep dive into specific modeling functions such as Calculated and Restricted Measures, Variables and Input Parameters in the next unit.

You will learn the different types of nodes you can use in Calculation Views and what possibilities each of them provides.

Business Example

You have created a dimension calculation view and understood the main concepts of node, data sources, output columns, semantics.

You are now ready to model Calculation Views that include measures. After creating a simple cube with Star Join Calculation View, you will explore additional features of cube calculation views.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Use measures in Calculation Views
- Explain the benefits of each type of node in Calculation Views
- Create and combine nodes in Calculation Views
- Use the features of Calculation Views to enhance the flexibility of this type of view

Measures in Calculation Views

Compared with attributes, measures bring an additional level of complexity in Calculation Views. In particular, you must define the relevant behavior of views when measures are aggregated based on a number of attributes.

> Note:
> Attributes are regular columns and, generally, they do not need to be aggregated. Aggregating data in a dimension calculation view would just be like a *Select Distinct* statement.

In SAP HANA, two types of graphical modeling objects handle measures:

- Calculation Views of type *Cube*
- Calculation Views of type *Cube with Star Join*

  This second type replaces the former Analytic Views, which are now deprecated, though still supported.

In a Calculation View, the measures can originate from several data sources, that is, several tables, information views, table functions, and so on.

For example, you can join two different tables and extract measures from both of them.

> Note:
> This is a key difference with the former Analytic Views, which support only one source for measure columns.

Standard Preview or Custom Query

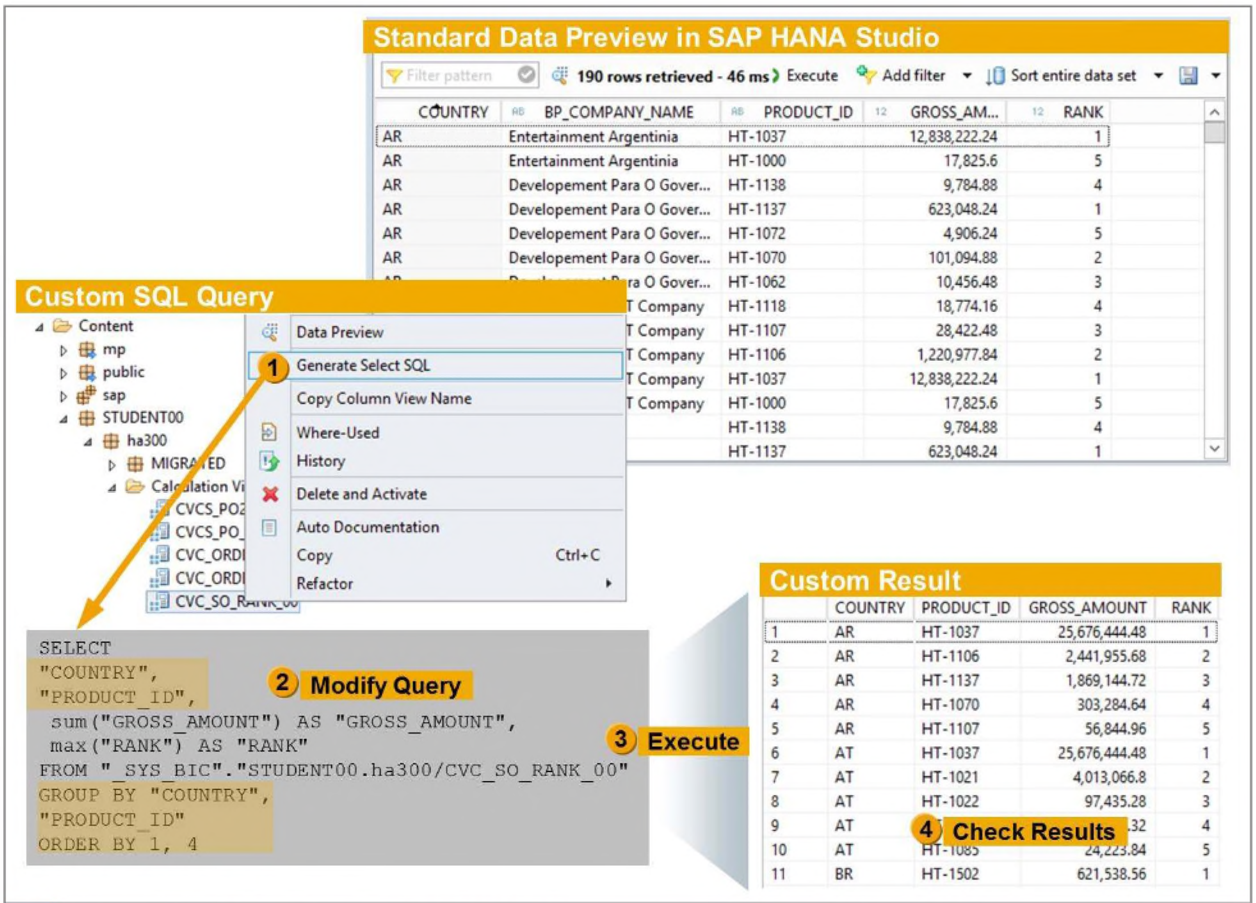


Figure 53: Standard Preview or Custom Query

To check the behavior of an Information View that has a complex calculation scenario, we recommend that you create custom Structured Query Language (SQL) queries on top of the view and test several scenarios. This is especially useful when some of the measures, or all of them, cannot be aggregated with the common Sum function, for example when working with ratios, averages and rack nodes.

This approach enables you to enhance the design of the view, if possible, or at least to understand how end users must query the view to get relevant results.

Overview of the Possible Nodes Types

A Calculation View can use as many nodes as you need. Each node has its own capabilities and behavior.

Node Type	Use Case
Projection	To filter data or obtain a subset of required columns from a data source
Aggregation	To summarize measures by grouping them together by attribute columns values
Join	To query data from two or more data sources
Union	To combine the data from two or more data sources
Star Join	To join attributes to the very last step of a *CUBE With Star Join* Calculation view
Rank	To order the data for a set of partition columns and select only the top 3/4/…/n elements

Figure 54: Types of Nodes in Graphical Calculation Views

Projection Node

A *Projection* node is typically used in the following scenarios:

- To filter measures based on attributes values.
- To extract only some columns from a data source.

Using a *Projection* node is sometimes a prerequisite to compute a calculated measure before an aggregation. Indeed, calculated columns in the *Aggregation* nodes of Calculation Views do not offer the option to execute the calculations before the aggregation.

Join Node

The *Join* node is used to join two (and only two) sources.

If you need to join more than two sources, you can stack several joins in the Scenario of your Calculation View.

For each *Join* node, you must define which columns of the two joined sources must participate in the join, as well as the join type, as discussed in the lesson *Connecting Tables*. You can also specify the cardinality (with the help of the join validation feature), but only if you are sure that the cardinality you provide corresponds to how the data is actually organized.

Spatial Join Type

The *Join* node type supports spatial joins, which means that you can now join tables that contain spatial data types in graphical calculation views. In some cases, graphical Spatial Join modeling avoids creating script-based calculation views or Table functions to perform spatial operations.

Enhancing Model Flexibility with Dynamic Join

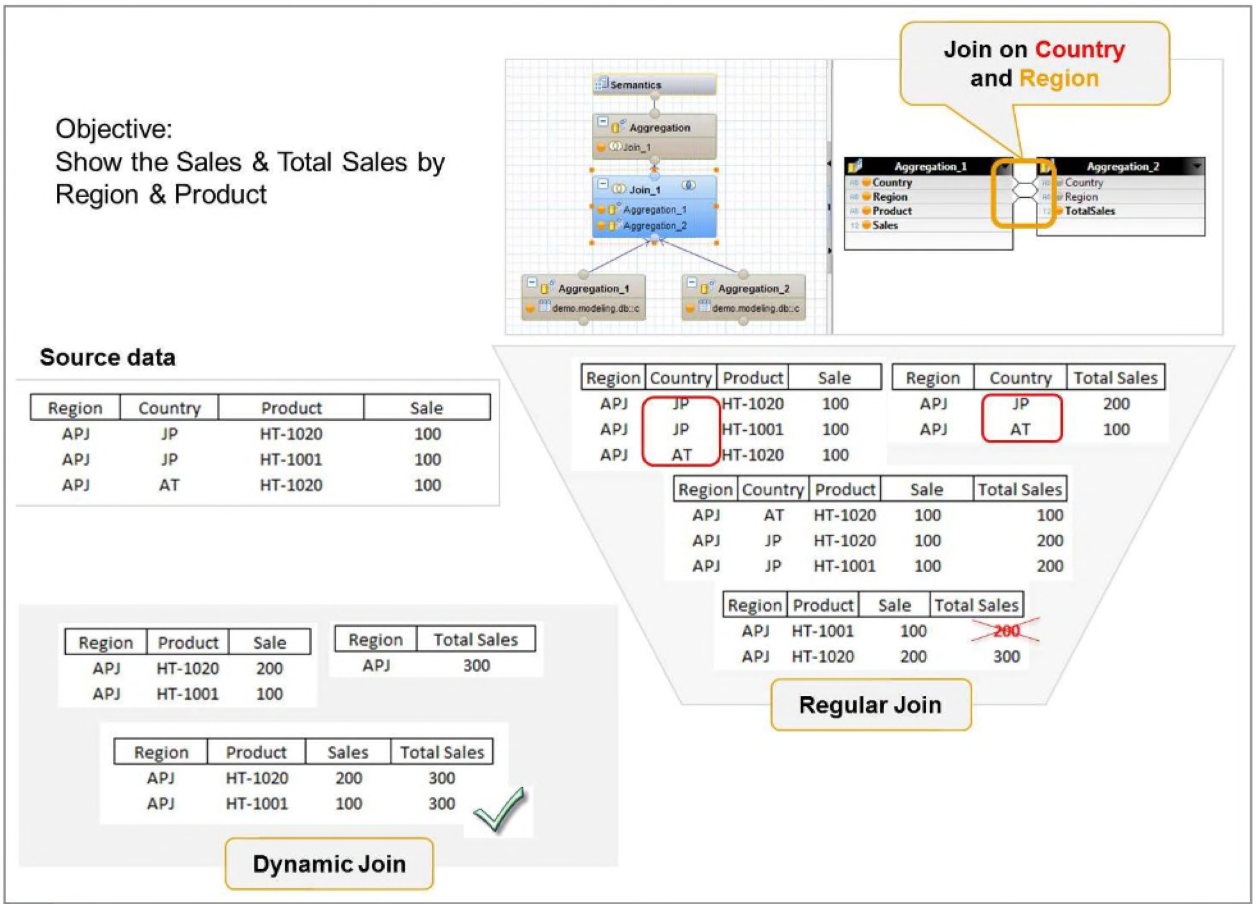


Figure 55: Dynamic Join

In some scenarios, you want to allow data analysis at different levels of granularity with the same Calculation View.

This is generally possible in an aggregation node when measures support the aggregation at different levels, that is, when it is possible to report measures grouped by one set of columns or another. For example, calculating the total sales by Country and Product in one case, and by Region and Product in another case.

The figure, Dynamic Join, shows an example of a more complex scenario, where you want to present two different measures side by side, either by Country, or Region:

- The sales by product
- The total sales (all products)

In this case, assuming that you model your calculation view with a regular join on Country and Region, you will get correct results if you analyze the data by Country, but the results will be inconsistent if you analyze the data by Region.

> Note:
> On further analysis of the example, you can see that the details of Total Sales by Region and Product is inconsistent for products that are not sold in all the Countries of the Region (HT-1001)... or (to be specific) in all the Countries of the Region that report Sales.

Benefits of a Dynamic Join

With a Dynamic Join, only the Join columns requested in the query are brought into context and play a part in the Join execution. As a consequence, the same Calculation View can be used for both purposes, that is, to analyze data by Country or by Region.

> Note:
> A Dynamic Join can be defined only with multi-column joins.

If we consider the behavior of the join from an aggregation perspective:

- In a regular (static) join, the aggregation is executed after the join.
- In a dynamic join, when a joined column is not requested by the client query, an aggregation is triggered to remove this column, and then the join is executed based only on the requested columns.

> Caution:
> With a dynamic join, if none of the joined columns are requested by the client query, you get a query run-time error.

Star Join Node

If your source data for a graphical calculation view is of a star schema type, you can create a calculation view of type *Cube with Star Join*.

A Star Join node enables you to join the fact data with the descriptive data. The input allowed to the Star Join include the lower node and all the relevant Calculation Views of type dimension.

This way, you are logically creating a star schema where the join is created from the central entity to the other entities.

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

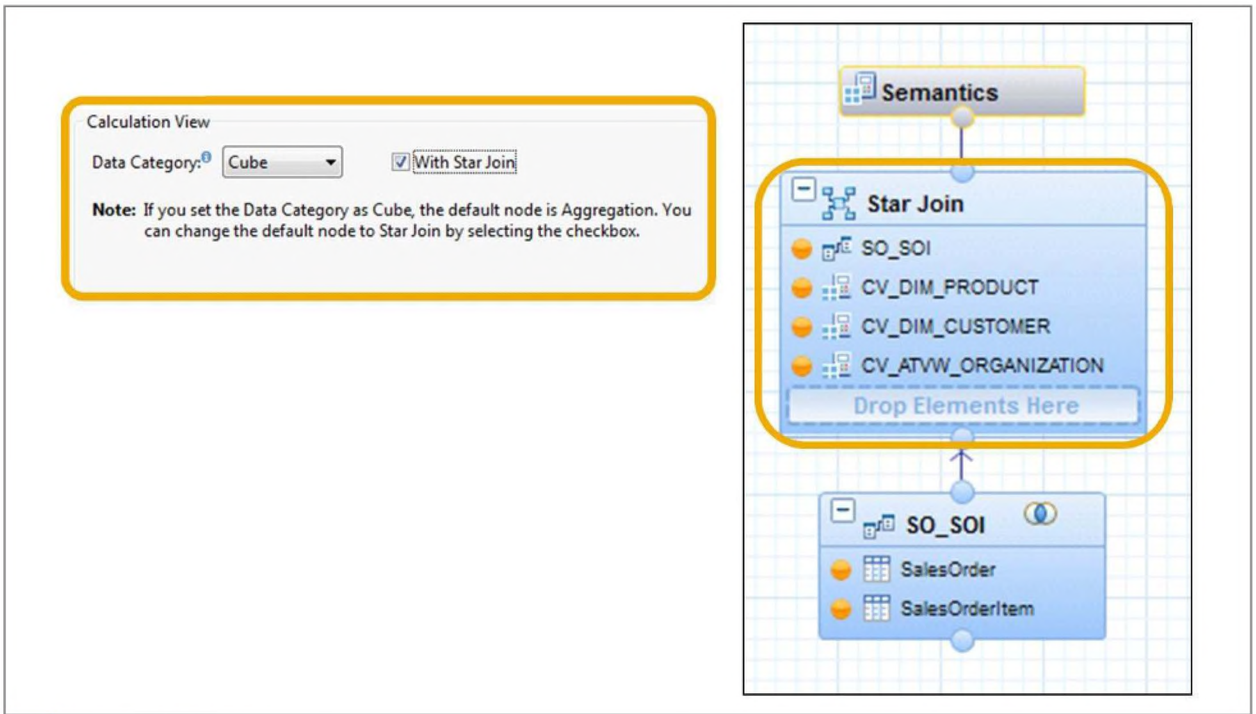


Figure 56: Star join

During deployment, the Star Join is always deployed with an aggregation node on top of it. The Star Join is deployed first with a series of joins, and then the aggregation logic is deployed. This allows the view to aggregate the measures dynamically, based on the attribute columns you select to include in the result set.

The Star Join in Calculation Views supports the *Referential* join type, which can speed up the execution of the Calculation View when you do not query any column from one or several of the dimension Calculation Views assigned to the Star Join node, as the join will not be executed (at least for the cardinalities *1..1* and *n..1*).

> Caution:
> As always, using a *Referential* join requires that the referential integrity of the joined tables is ensured.

Unit 1
Exercise 3

Create a CUBE with Star Join Calculation View

Exercise Objectives

After completing this exercise, you will be able to:

- Create a Star Join Calculation View
- Reuse existing Dimension Calculation Views, including the one you created for Business Partners.

Business Example

You need to report on purchase order data. To meet the business requirement, you have decided to create a Star Join Calculation View.

> Note:
> The Fact table will be based on two sources from the SAP Enterprise Performance Management (EPM) Model: the Purchase Order headers, and the Purchase Order Items.
>
> In this exercise, when the values include ##, replace the characters with the number that your instructor assigned you.

1. In your *STUDENT##* package, create a new Calculation View with the following properties:

Field	Value
Name	**CVCS_PO_##** (## is your student number)
Description	**Purchase Orders**
Subtype	*Standard*
Calculation View Type	*Graphical*
Category	*CUBE*
With Star Join	[Selected]

2. Add a Join node to the *Scenario* pane, and add the following data sources (tables from the *EPM_MODEL* schema) to the new node:

   - Table *EPM_MODEL.SNWD_PO* (Purchase order document header)
   - Table *EPM_MODEL.SNWD_PO_I* (Purchase order line items)

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

> Note:
> The new Join node will be automatically called *Join_1*

3. In the *Details* pane, join the data sources of node *Join_1* as follows, for each join use a *Referential* join type and a *1..n* cardinality.

   The table *SNWD_PO* is considered as the left table, and the table *SNWD_PO_I* as the right table.

   - Join the field *CLIENT* of the left table to the field *CLIENT* of the right table.
   - Join the field *NODE_KEY* of the left table to the field *PARENT_KEY* of the right table.

> Hint:
> To check the join type and cardinality, in the *Edit Join* screen, choose *Validate Join*.

4. In the *Details* pane for node *Join_1*, add the following columns to the output:

Source table	Columns
SNWD_PO	• CLIENT • PO_ID • PARTNER_GUID
SNWD_PO_I	• PO_ITEM_POS • PRODUCT_GUID • CURRENCY_CODE • GROSS_AMOUNT • TAX_AMOUNT • NET_AMOUNT

5. Define the *Join_1* node as the data source for the *Star Join* node.
6. Add the following Dimension Calculation Views to the *Star Join* node:

Source Package	Dimension Calculation View
STUDENT##	CVD_BP_## (Business partners)
training.ha300	CVD_PD (Products)

7. In the *Star Join* node, join the *Join_1* table with the two Dimension Calculation Views as shown in the following table, using a *Referential* join type and a *n..1* cardinality.

   Join_1 is considered as the left table, and the Dimension Calculation Views as the right tables.

Left Table Column	Right Tables Column
CLIENT	CVD_BP_##.CLIENT
PARTNER_GUID	CVD_BP_##.NODE_KEY
CLIENT	CVD_PD.CLIENT
PRODUCT_GUID	CVD_PD.NODE_KEY

8. In the *Details* pane for the *Star Join* node, add all columns from the *Join_1* table to the output, except the *PARTNER_GUID* and *PRODUCT_GUID*.

> Note:
> We make the assumption that these columns, used only to join the fact table to the Dimension views, are not required by the end user of the *CVCS_PO_##* View.

9. In the *Semantics* node, assign the relevant *Type* property to all the local columns. The three columns with amounts are Measures, all the others are Attributes.

> Hint:
> You can use *Auto Assign* in the top-right corner of the *Columns* tab, and check the corresponding result.

10. Hide the *CLIENT* and *NODE_KEY* columns (shared attributes) from the Dimension Calculation Views *CVD_BP_##* and *CVD_PD*.

11. Save and activate the Calculation View.

> Note:
> The *CLIENT* and *NODE_KEY* columns exist in both Dimension Calculation Views, and the system cannot accept duplicate column names.
>
> Accept the proposed alias names for the duplicate entities.

    Please check the activation status after activation.

12. Preview the data.

Unit 1
Solution 3

Create a CUBE with Star Join Calculation View

Exercise Objectives

After completing this exercise, you will be able to:

- Create a Star Join Calculation View
- Reuse existing Dimension Calculation Views, including the one you created for Business Partners.

Business Example

You need to report on purchase order data. To meet the business requirement, you have decided to create a Star Join Calculation View.

> Note:
> The Fact table will be based on two sources from the SAP Enterprise Performance Management (EPM) Model: the Purchase Order headers, and the Purchase Order Items.
>
> In this exercise, when the values include ##, replace the characters with the number that your instructor assigned you.

1. In your *STUDENT##* package, create a new Calculation View with the following properties:

Field	Value
Name	**CVCS_PO_##** (## is your student number)
Description	**Purchase Orders**
Subtype	*Standard*
Calculation View Type	*Graphical*
Category	*CUBE*
With Star Join	[Selected]

a) In the *Content* folder, right-click your *STUDENT##* package and choose *New → Calculation View*.

b) Enter the Calculation View name and other properties as shown in the table.

c) Choose *Finish*.
The Calculation View graphical editor opens on the right of your screen.

2. Add a Join node to the *Scenario* pane, and add the following data sources (tables from the *EPM_MODEL* schema) to the new node:

   - Table *EPM_MODEL.SNWD_PO* (Purchase order document header)
   - Table *EPM_MODEL.SNWD_PO_I* (Purchase order line items)

   > Note:
   > The new Join node will be automatically called *Join_1*

   a) In the *Scenario* pane, choose the + sign on the right of the *Join_1* node.

   b) In the search field, enter **SNWD_PO**.

   c) On your keyboard, press Ctrl, and select the tables.

   d) Choose *OK*.

3. In the *Details* pane, join the data sources of node *Join_1* as follows, for each join use a *Referential* join type and a *1..n* cardinality.

   The table *SNWD_PO* is considered as the left table, and the table *SNWD_PO_I* as the right table.

   - Join the field *CLIENT* of the left table to the field *CLIENT* of the right table.
   - Join the field *NODE_KEY* of the left table to the field *PARENT_KEY* of the right table.

   > Hint:
   > To check the join type and cardinality, in the *Edit Join* screen, choose *Validate Join*.

   a) In the *Details* pane, drag the *SNWD_PO.CLIENT* field to the *SNWD_PO_I.CLIENT* field, and then drag the *SNWD_PO.NODE_KEY* field to the *SNWD_PO_I.PARENT_KEY* field.

   > Caution:
   > For the second join, make sure you actually join the PARENT_KEY of the right table (NOT the NODE_KEY).

   b) To edit the join properties, double-click the new connector.

   c) In the *Join Type* dropdown list, choose *Referential*.

   d) In the *Cardinality* dropdown list, choose *1..n*.

   e) Optionally, choose *Join Validation* to check the join properties.

   f) Choose *OK*.

4. In the *Details* pane for node *Join_1*, add the following columns to the output:

Source table	Columns
SNWD_PO	• CLIENT • PO_ID • PARTNER_GUID
SNWD_PO_I	• PO_ITEM_POS • PRODUCT_GUID • CURRENCY_CODE • GROSS_AMOUNT • TAX_AMOUNT • NET_AMOUNT

a) In the *EPM_MODEL.SNWD_PO_I* table, right-click the *CLIENT* field and choose *Add To Output*.

> Hint:
> Alternatively, you can double-click the field name or choose toggle on the left of a table field.

b) Repeat the previous step for the other fields to be included in the output.

5. Define the *Join_1* node as the data source for the *Star Join* node.

a) In the *Scenario* pane, draw a connector from the dot on top of the *Join_1* node to the dot at the bottom of the *Star Join* node.

6. Add the following Dimension Calculation Views to the *Star Join* node:

Source Package	Dimension Calculation View
STUDENT##	CVD_BP_## (Business partners)
training.ha300	CVD_PD (Products)

a) In the *Scenario* pane, choose the + sign on the right of the *Star Join* node.

b) In the search field, enter **CVD_BP**.

c) Select the *CVD_BP_##* view.

d) Choose *OK*.

e) For the second view, repeat the previous steps with the search criteria **CVD_PD**.

7. In the *Star Join* node, join the *Join_1* table with the two Dimension Calculation Views as shown in the following table, using a *Referential* join type and a *n..1* cardinality.

   Join_1 is considered as the left table, and the Dimension Calculation Views as the right tables.

Left Table Column	Right Tables Column
CLIENT	CVD_BP_##.CLIENT
PARTNER_GUID	CVD_BP_##.NODE_KEY
CLIENT	CVD_PD.CLIENT
PRODUCT_GUID	CVD_PD.NODE_KEY

   a) In the *Details* pane of the *Star Join* node, drag the field *CLIENT* from the *Join_1* table to the field *CLIENT* of the Calculation View *CVD_BP_##*.

   b) Drag the field *PARTNER_GUID* from the *Join_1* table to the field *NODE_KEY* of the Calculation View *CVD_BP_##*.

   c) To edit the join properties, double-click the new connector of one join.

   d) In the *Join Type* dropdown list, choose *Referential*.

   e) In the *Cardinality* dropdown list, choose *n..1*.

   f) Choose *OK*.

   g) Repeat the previous steps for the other join to the Dimension Calculation View *CVD_PD*.

8. In the *Details* pane for the *Star Join* node, add all columns from the *Join_1* table to the output, except the *PARTNER_GUID* and *PRODUCT_GUID*.

   > Note:
   > We make the assumption that these columns, used only to join the fact table to the Dimension views, are not required by the end user of the *CVCS_PO_##* View.

   a) In the *Join_1* table, right-click the *CLIENT* field and choose *Add To Output*.

   b) Repeat the previous step for all the fields to be included in the output. Do NOT include the *PARTNER_GUID* and *PRODUCT_GUID* columns.

9. In the *Semantics* node, assign the relevant *Type* property to all the local columns. The three columns with amounts are Measures, all the others are Attributes.

   > Hint:
   > You can use *Auto Assign* in the top-right corner of the *Columns* tab, and check the corresponding result.

   a) Choose *Auto Assign* in the top-right corner of the *Columns* tab, and check the corresponding result.

> Note:
> Alternatively, you can select several columns from the list. On your keyboard, press Ctrl, or Shift, select the columns, and choose *Mark as Attribute* or *Mark as Measure*.

10. Hide the *CLIENT* and *NODE_KEY* columns (shared attributes) from the Dimension Calculation Views *CVD_BP_##* and *CVD_PD*.
    a) Select the *Columns* tab of the *Semantics* node.
    b) In the *Shared* columns list, select the *Hide* checkbox of the two *CLIENT* columns and the two *NODE_KEY* columns.
11. Save and activate the Calculation View.

> Note:
> The *CLIENT* and *NODE_KEY* columns exist in both Dimension Calculation Views, and the system cannot accept duplicate column names.
>
> Accept the proposed alias names for the duplicate entities.

Please check the activation status after activation.

a) Choose *Save and Activate*.
b) Accept the proposed aliases for the duplicate entities.
c) Choose *Save and Activate*.

12. Preview the data.
    a) Choose *Data Preview*, then *Open in Data Preview Editor* and select the *Raw Data* tab.
    b) Review the content.
    c) Close the data preview.

Union Node

- If you want to combine multiple result sets with identical structures into one result set, you can use a union node
- A mapping of the sources to the target is required and will allow you to adapt structural differences
- This can be done via a drag and drop interface.

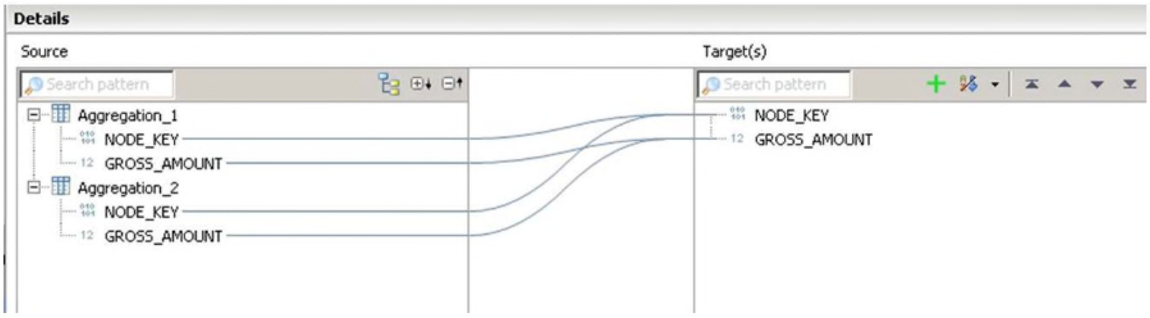


Figure 57: Union Node

A *Union* node is used to combine two or more data sets to a common set of columns.

Depending on how different the column names are from the data source, you can map the columns automatically by name (columns with identical names will be mapped automatically) or adjust the mapping manually.

Standard Union

Depending on the requirement, you can use one of the following approaches:

- A standard union
- A union with constant values

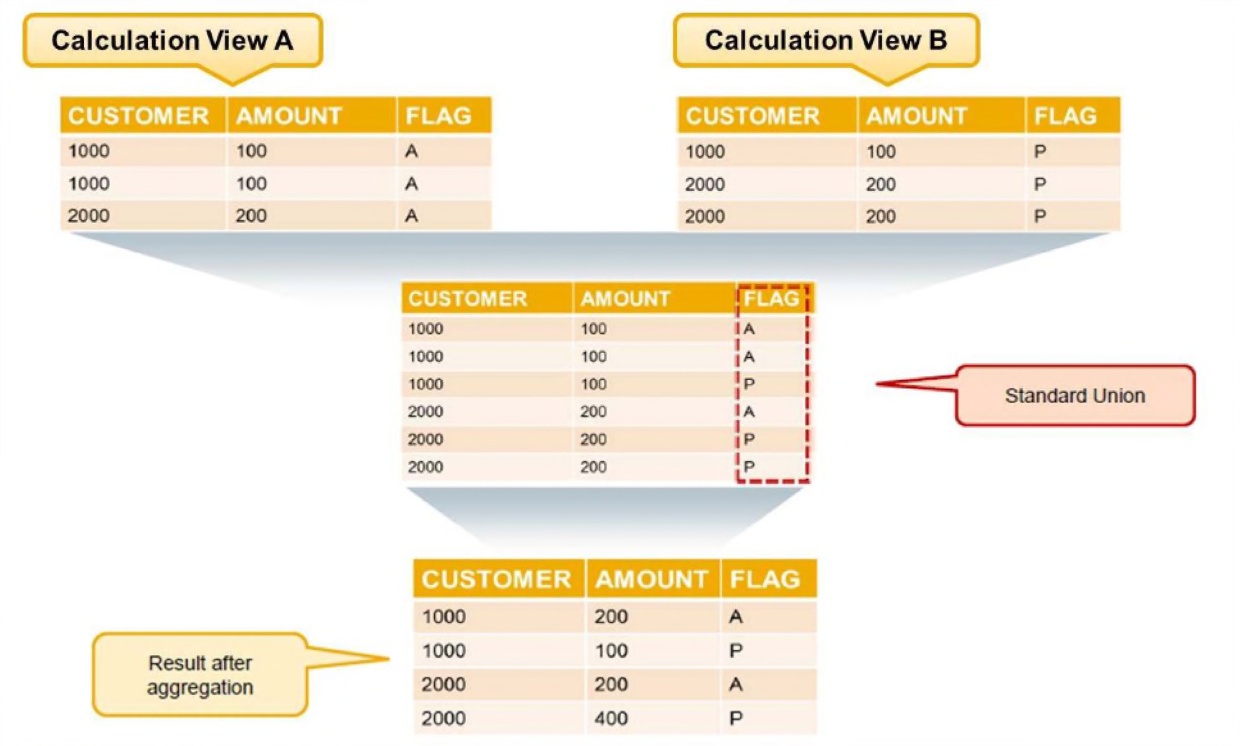


Figure 58: Standard Union

Union with Constant Values

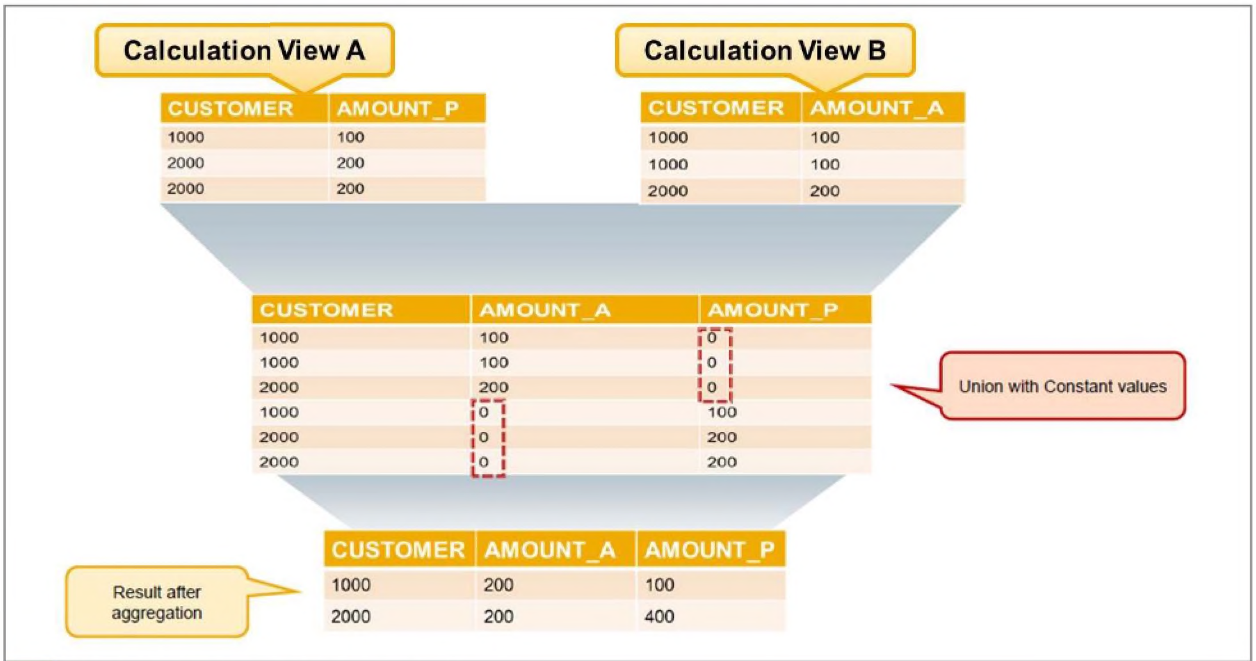


Figure 59: Union with Constant Values

The choice between a standard union and a union with constant values depends on the data you are using and the way you want the end users to report on data.

If it is more beneficial to present different measures in different columns, you can use union with constant values. On the contrary, if it is easier to present measures in a single column and differentiate them with an attribute, use the standard union.

Unmapped Columns in a Union Node

- There could be instances when a union needs to be performed where the sources have a different number of columns.
- You can then set a constant value for the source columns that do not have the target column.
- To set the constant value, right click the target column and choose “Manage Mappings”.

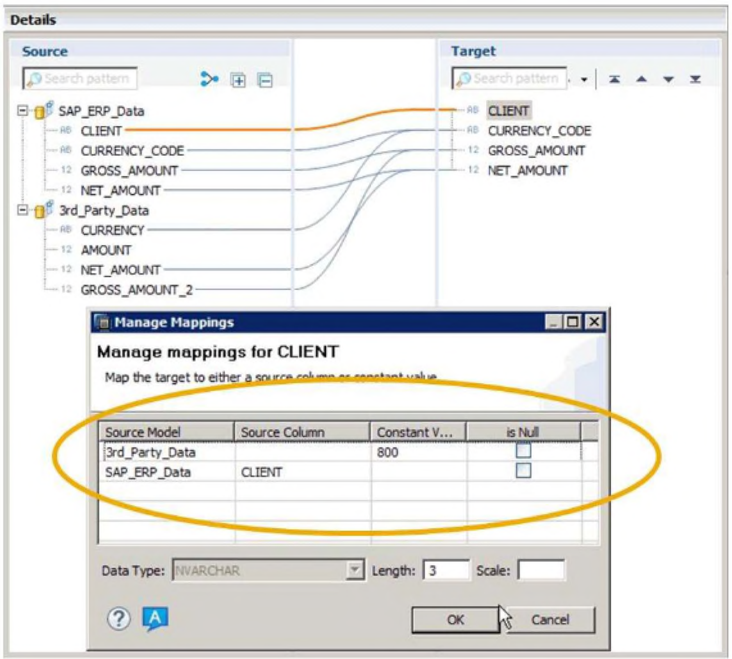


Figure 60: Unmapped Columns in a Union Node

Union Node Pruning

SAP HANA service pack stack (SPS) 11 provides a possibility to optimize the execution of union nodes, by specifying pruning conditions in a dedicated table (Pruning Configuration Table).

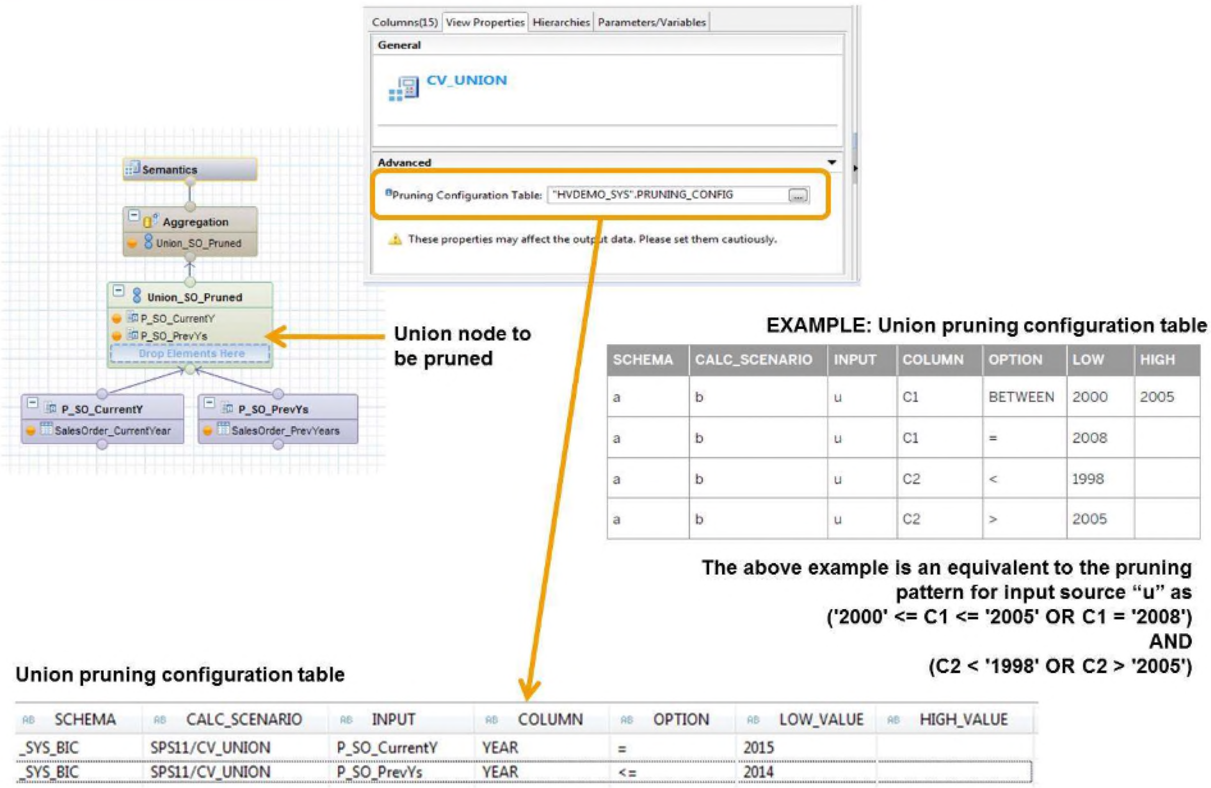


Figure 61: Union Node Pruning

The pruning is defined for a specific Calculation view (CALC_SCENARIO). It specifies the pruning conditions to be applied to one or several nodes or columns for this view.

- The possible operators are =, <, >, <=, >= and BETWEEN.
- The BETWEEN operator refers to a closed interval (including the LOW and HIGH values).
- If several pruning conditions exist for the same column of a view, they are combined with *OR*.
- On different columns, the resulting conditions are combined with *AND*.

Unit 1
Exercise 4

Combine Two Data Sources with a Union Node

Exercise Objectives

After completing this exercise, you will be able to:

- Create calculated columns in Calculation Views
- Use aggregation nodes in Calculation Views
- Work with data types in Calculation Views
- Use union nodes in Calculation Views

Business Example

A company wants to analyze its sales data, which is available in SAP HANA but in two different tables, as it is extracted from two different ERP systems, one from SAP and the other from a third party vendor.

You decided to use a calculation view for this new report, where the union node allows you to combine the data from the two data tables together.

To enable the union, you must harmonize the data types between the tables.

Task 1: Create a Calculation View to Support the Business Requirements

1. Create a new Calculation View using the following details:

Property	Value
Name	*CVC_SO_##* (## is your group number)
Description	**Summary of Sales Orders from two Source Systems**
Type	*Graphical*
Data Category	*CUBE*
With Star Join	[Deselected]

Task 2: Add the Sales Data from the SAP Source

1. In the scenario pane, create a new aggregation node (in addition to the node automatically generated) at the bottom left of the view.

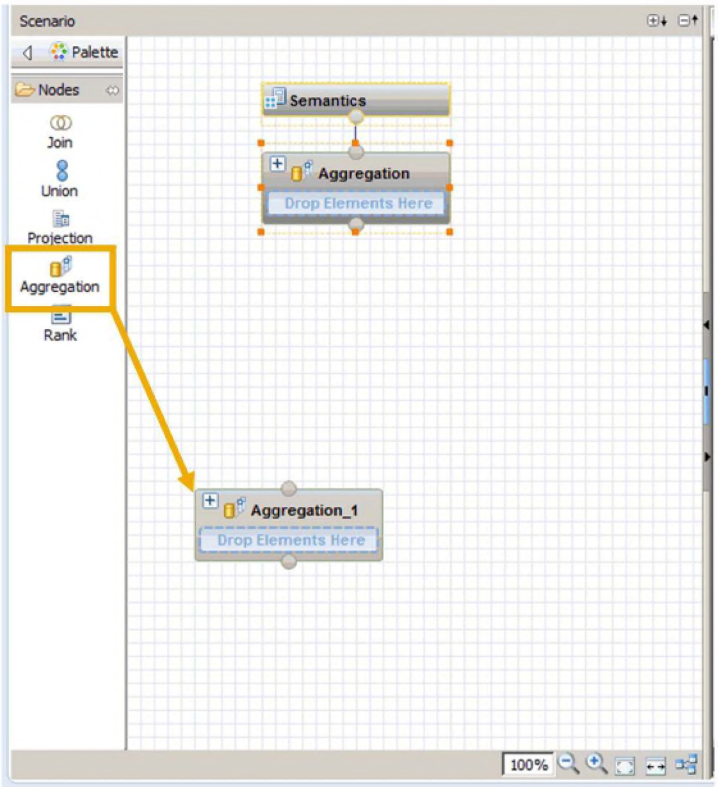


Figure 62: Creating an Additional Aggregation Node

2. Rename the new aggregation node **`SAP_ERP_Data`**.
3. Add the table *SNWD_SO* from the schema *EPM_MODEL* to the *SAP_ERP_Data* node.
4. Add the following attributes from the table *SNWD_SO* to the output:

   CLIENT, *CURRENCY_CODE*

> Caution:
> Do not add these columns as aggregated columns.
>
> Aggregated columns are used for measures.

5. Add the following measures (as aggregated columns) from the table *SNWD_SO* to the output.

   GROSS_AMOUNT, *NET_AMOUNT*

Task 3: Add the Sales Data from the Third Party Source

1. Create another aggregation node **`3rd_Party_Data`** and place it to the right of the *SAP_ERP_Data* node. The data source for the new node is the table *SALES_DATA* from the *TRAINING* schema.
2. Add the attribute *CURRENCY* from the *3rd_Party_Data* aggregation node to the output.
3. Add the measure *AMOUNT* as an aggregated column.

Task 4: Create Calculated Columns

Third party data does not contain the net amount, so this measure must be calculated.

Additionally, the columns *GROSS_AMOUNT* and *NET_AMOUNT* from the SAP source are using data type *Decimal (15,2)*, and you want to ensure that the gross and net amounts from third party data are using the same data type.

1. In the *3rd_Party_Data* aggregation node, add a calculated column using the following details:

Field	Value
Name	**NET_AMOUNT**
Data Type	*Decimal*
Length	**15**
Scale	**2**
Expression	**"AMOUNT" * 0.9**

2. Add another calculated column using the following details:

Field	Value
Name	**GROSS_AMOUNT_2**
Data Type	*Decimal*
Length	**15**
Scale	**2**
Expression	**"AMOUNT"**

> Note:
> This calculated column is added for the sole purpose of data type conversion.

Task 5: Combine the Two Data Sources

Use a Union node to combine the result set from the two lower aggregation nodes.

1. In the *Scenario* pane, add a *Union* node above the *SAP_ERP_Data* and *3rd_Party_Data* aggregation nodes and connect these aggregation nodes to the new Union node.

2. In the *Union_1* node, specify how the columns from the two data sources must be combined.

   All the columns from the *SAP_ERP_Data* must be mapped to the target with the same name.

   For the columns of the *3rd_Party_Data* source, define the mapping as follows:

Source column	Target column
CURRENCY	CURRENCY_CODE
AMOUNT	[no mapping]
NET_AMOUNT	NET_AMOUNT
GROSS_AMOUNT_2	GROSS_AMOUNT

3. The data from the third party system does not contain the *CLIENT* column. To ensure consistent data, assign a constant value for the *CLIENT* target column to all the rows from the *3rd_Party_Data* source.

   This value must be set to **800**, which is the value assigned to the SAP system data.

Hint:
To manage a column mapping, right-click the *CLIENT* target column and choose *Manage Mappings*.

Caution:
To prevent an issue that might occur when validating the model, you must remove the mapping of column *CLIENT* (source *SAP_ERP_Data*) and define it again:

- Right-click the *CLIENT* column in source *SAP_ERP_Data* and choose *Remove Mapping*.
- Right-click the same column again and choose *Map to Target*. Choose the target column *CLIENT* and choose *OK*.

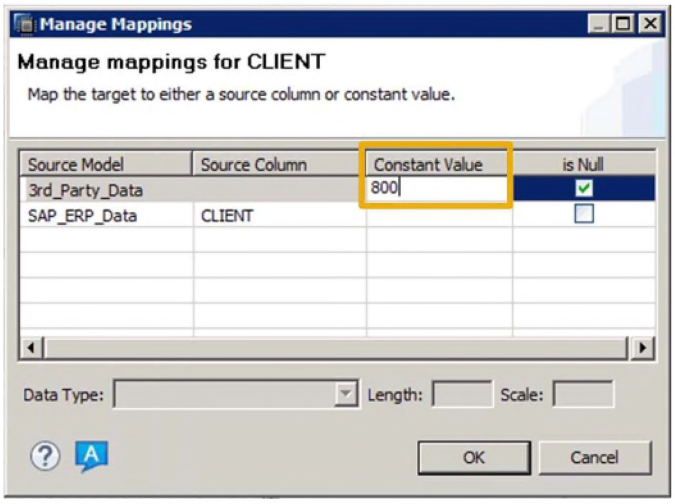


Figure 66: Defining a Mapping with Constant Value

Task 6: Finalize the Calculation View

1. In the scenario, connect the *Union_1* node to the uppermost aggregation node.
2. In the *Aggregation* node, add the attributes *CLIENT* and *CURRENCY_CODE* to the output (non-aggregated columns), and the measures *GROSS_AMOUNT* and *NET_AMOUNT* as aggregated columns.
3. Save and activate the calculation view.
4. Preview the data of your Calculation View.
5. What is the value of GROSS_AMOUNT for CURRENCY_CODE: USD?
6. Close the data preview.

Note:
An intermediate solution object is available at this stage:
STUDENT00 ha300 CVC_SO_00_STAGE1

Task 7: Replace the Data Source for the 3rd_Party_Data Aggregation Node

You have just finished designing your calculation view, when you are informed that the third-party data source you took was not the right one. You must replace the *SALES_DATA* table with the *SALES_DATA_NEW* table in your model.

1. Open the calculation view *CVC_SO_##*, if needed.
2. In the Catalog, open the table *SALES_DATA_NEW* and compare the name of its columns with the columns of the table *SALES_DATA* that appear in the *3rd_Party_Data* node of the calculation view (in the *Details* pane).
3. Replace the data source of the *3rd_Party_Data* node with the table *SALES_DATA_NEW* and to avoid any inconsistency in the calculation view, make sure that you correctly map the changed column names.

Hint:
Take care not to click *Finish* just after selecting the new data source; otherwise, the table will be replaced by the new one without any column mapping.

Additionally, at the *Map Columns* step, note that you must first delete the inconsistent mapping by right-clicking the *AMOUNT* column in the target.

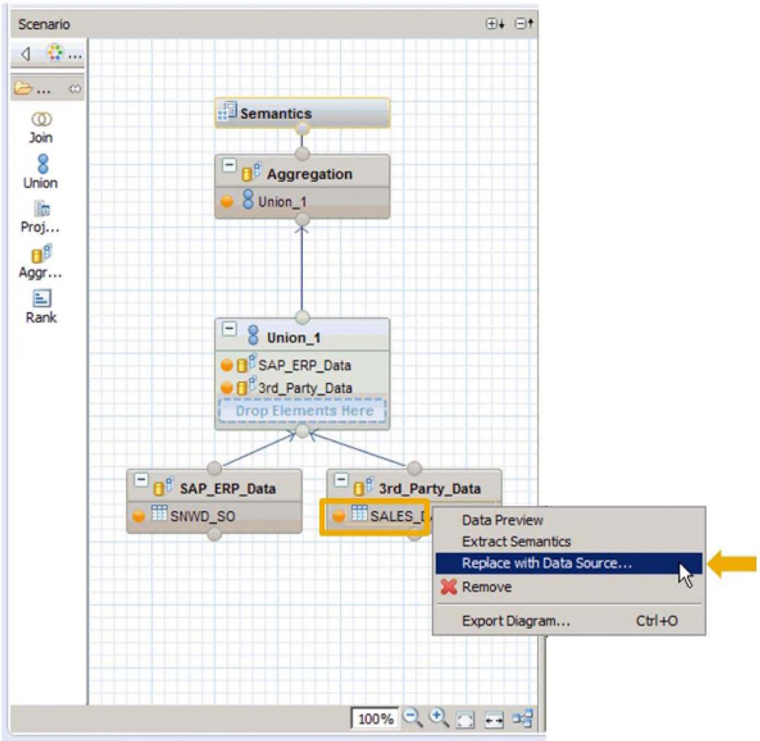


Figure 68: Replacing a Data Source

4. Check the consistency of the Calculation view. In particular, in the output of the *3rd_Party_Data* node, check that the *AMOUNT* column has the correct mapping to the column *SALES_DATA_NEW.GROSS_AMOUNT*.

5. Activate the Calculation View.

6. Preview the Data.

Unit 1
Solution 4

Combine Two Data Sources with a Union Node

Exercise Objectives

After completing this exercise, you will be able to:

- Create calculated columns in Calculation Views
- Use aggregation nodes in Calculation Views
- Work with data types in Calculation Views
- Use union nodes in Calculation Views

Business Example

A company wants to analyze its sales data, which is available in SAP HANA but in two different tables, as it is extracted from two different ERP systems, one from SAP and the other from a third party vendor.

You decided to use a calculation view for this new report, where the union node allows you to combine the data from the two data tables together.

To enable the union, you must harmonize the data types between the tables.

Task 1: Create a Calculation View to Support the Business Requirements

1. Create a new Calculation View using the following details:

Property	Value
Name	*CVC_SO_##* (## is your group number)
Description	`Summary of Sales Orders from two Source Systems`
Type	*Graphical*
Data Category	*CUBE*
With Star Join	[Deselected]

a) In the content node of the *Systems* view, right-click on your package and select *New → Calculation View*.

b) Enter the details from the table.

c) Choose *Finish*.

Task 2: Add the Sales Data from the SAP Source

1. In the scenario pane, create a new aggregation node (in addition to the node automatically generated) at the bottom left of the view.

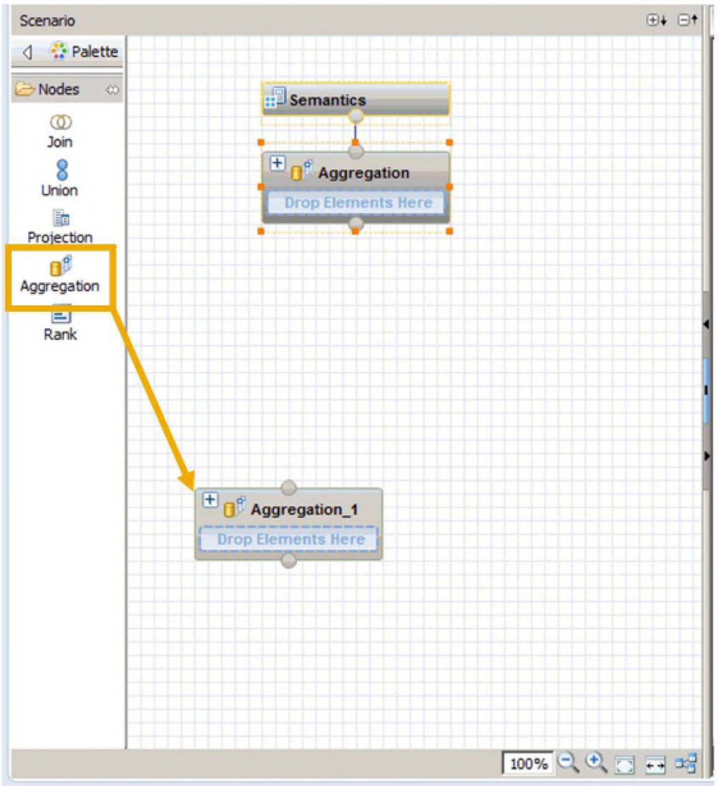


Figure 62: Creating an Additional Aggregation Node

a) Drag the *Aggregation node* from the Palette to the bottom left of *Scenario* pane.

2. Rename the new aggregation node **SAP_ERP_Data**.

   a) Right-click the new aggregation node and choose *Rename*.

   b) Enter **SAP_ERP_Data**.

   c) Choose *OK*.

3. Add the table *SNWD_SO* from the schema *EPM_MODEL* to the *SAP_ERP_Data* node.

   a) From the catalog node, drag the table *EPM_MODEL.SNWD_SO* to the *SAP_ERP_Data* aggregation node.

> Hint:
> You can also right-click the aggregation node, choose *Add Objects*, and use the search feature to find the specified table.

4. Add the following attributes from the table *SNWD_SO* to the output:

   CLIENT, *CURRENCY_CODE*

> Caution:
> Do not add these columns as aggregated columns.
>
> Aggregated columns are used for measures.

   a) In the *SAP_ERP_Data* aggregation node, right-click the specified columns and choose *Add To Output*.

5. Add the following measures (as aggregated columns) from the table *SNWD_SO* to the output.

   GROSS_AMOUNT, *NET_AMOUNT*

   a) In the *SAP_ERP_Data* aggregation node, right-click the specified columns and choose *Add As Aggregated Column*.

Task 3: Add the Sales Data from the Third Party Source

1. Create another aggregation node **3rd_Party_Data** and place it to the right of the *SAP_ERP_Data* node. The data source for the new node is the table *SALES_DATA* from the *TRAINING* schema.

   a) Drag the *Aggregation node* from the Palette to the right of the *SAP_ERP_Data* aggregation node.

   b) Right-click the new aggregation node, choose *Rename*, enter **3rd_Party_Data**, and choose *OK*.

   c) From the catalog node, drag the table *TRAINING.SALES_DATA* to the new aggregation node.

2. Add the attribute *CURRENCY* from the *3rd_Party_Data* aggregation node to the output.

   a) In the details pane, right-click the *CURRENCY* column and choose *Add To Output*.

3. Add the measure *AMOUNT* as an aggregated column.

   a) Right-click the *AMOUNT* column and choose *Add As Aggregated Column*.

Task 4: Create Calculated Columns

Third party data does not contain the net amount, so this measure must be calculated.

Additionally, the columns *GROSS_AMOUNT* and *NET_AMOUNT* from the SAP source are using data type *Decimal (15,2)*, and you want to ensure that the gross and net amounts from third party data are using the same data type.

1. In the *3rd_Party_Data* aggregation node, add a calculated column using the following details:

Field	Value
Name	**NET_AMOUNT**
Data Type	*Decimal*
Length	**15**
Scale	**2**
Expression	**"AMOUNT" * 0.9**

   a) Select the *3rd_Party_Data* aggregation node.

      In the *Output* pane, right-click the *Calculated Columns* folder and choose *New*.

   b) Enter the details as listed in the table.

2. Add another calculated column using the following details:

Field	Value
Name	**GROSS_AMOUNT_2**

Field	Value
Data Type	*Decimal*
Length	**15**
Scale	**2**
Expression	**"AMOUNT"**

Note:
This calculated column is added for the sole purpose of data type conversion.

a) In the *Output* pane, right-click the *Calculated Columns* folder and choose *New*.

b) Enter the details as listed in the table.

c) Choose *OK*.

Task 5: Combine the Two Data Sources

Use a Union node to combine the result set from the two lower aggregation nodes.

1. In the *Scenario* pane, add a *Union* node above the *SAP_ERP_Data* and *3rd_Party_Data* aggregation nodes and connect these aggregation nodes to the new Union node.

   a) Drag the *Union* node from the Palette to the area above the two aggregation nodes in the *Scenario* pane.

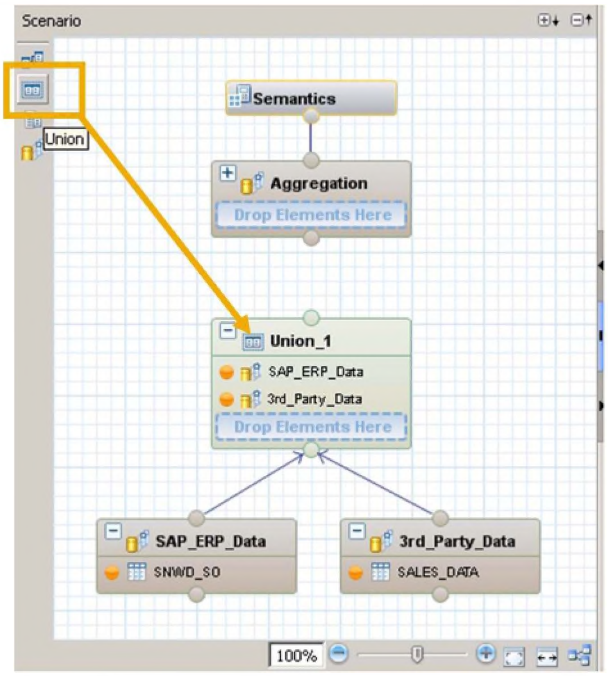


Figure 63: Adding a Union Node

   b) To connect the nodes, with your mouse, draw a connector between the round anchors of the nodes.

2. In the *Union_1* node, specify how the columns from the two data sources must be combined.

All the columns from the *SAP_ERP_Data* must be mapped to the target with the same name.

For the columns of the *3rd_Party_Data* source, define the mapping as follows:

Source column	Target column
CURRENCY	CURRENCY_CODE
AMOUNT	[no mapping]
NET_AMOUNT	NET_AMOUNT
GROSS_AMOUNT_2	GROSS_AMOUNT

a) Select the *Union_1* node.

b) In the *Source* list of the *Details* pane, right-click the *SAP_ERP_Data* and choose *Add to Target*.

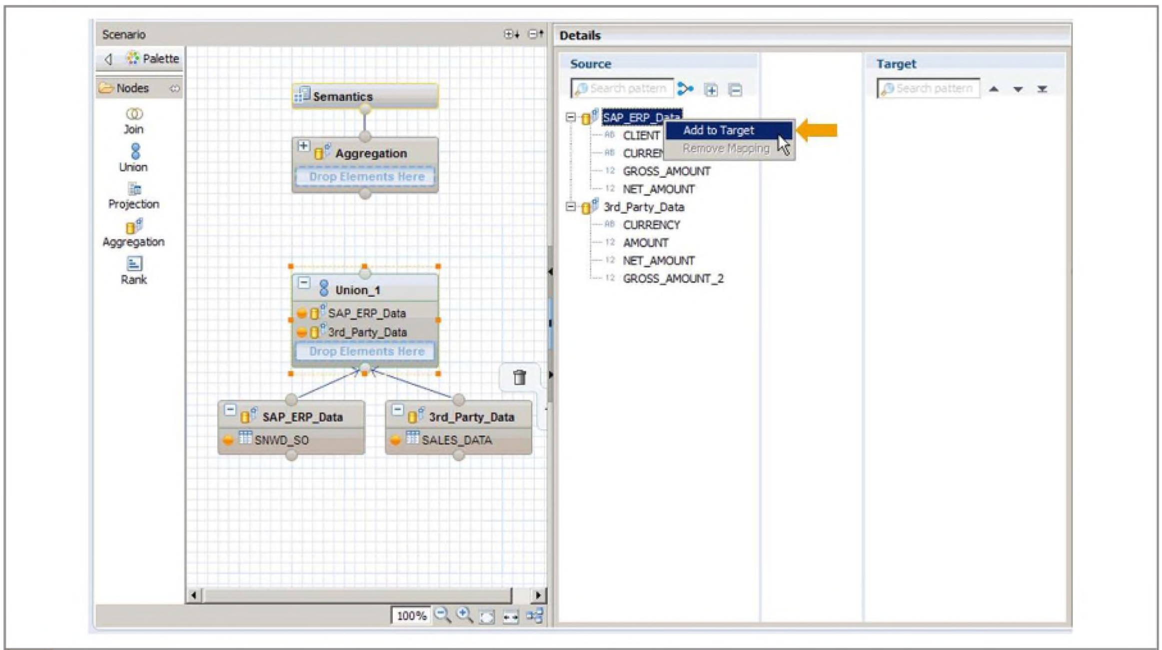


Figure 64: Defining Mappings for Source SAP_ERP_Data

c) For the source *3rd_Party_Data*, drag the *CURRENCY* column to the *CURRENCY_CODE* column of the target.

> Hint:
> You can also right-click the *CURRENCY* column, choose *Map to Target*, and in the *Target Column* dropdown list, select *CURRENCY_CODE* and choose *Create*.

d) Repeat the previous step to map the two other columns.

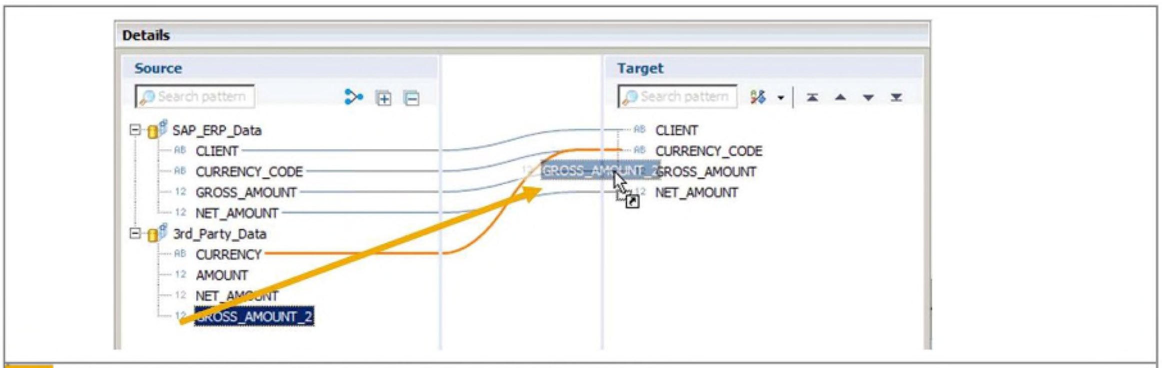


Figure 65: Defining Mappings for Source Third Part Data

3. The data from the third party system does not contain the *CLIENT* column. To ensure consistent data, assign a constant value for the *CLIENT* target column to all the rows from the *3rd_Party_Data* source.

   This value must be set to **800**, which is the value assigned to the SAP system data.

> Hint:
> To manage a column mapping, right-click the *CLIENT* target column and choose *Manage Mappings*.

> Caution:
> To prevent an issue that might occur when validating the model, you must remove the mapping of column *CLIENT* (source *SAP_ERP_Data*) and define it again:
>
> - Right-click the *CLIENT* column in source *SAP_ERP_Data* and choose *Remove Mapping*.
> - Right-click the same column again and choose *Map to Target*. Choose the target column *CLIENT* and choose *OK*.

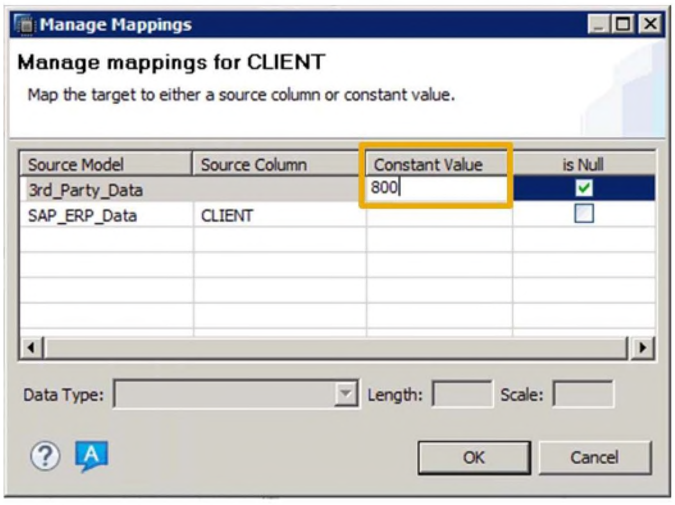


Figure 66: Defining a Mapping with Constant Value

a) In the Target, right-click the *CLIENT* column and choose *Manage Mappings*.
   The *Manage Mappings* dialog appears.

b) In the *Manage Mappings* dialog, for the *Source Model 3rd_Party_Data* entry, in the *Constant Value* column, enter `800`.

c) Choose *OK*.

d) Right-click the *CLIENT* column in source *SAP_ERP_Data* and choose *Remove Mapping*.

e) Right-click the same column again and choose *Map to Target*.

f) Choose the target column *CLIENT* and click *OK*.

Task 6: Finalize the Calculation View

1. In the scenario, connect the *Union_1* node to the uppermost aggregation node.

   a) To connect the nodes, with your mouse, draw a connector between the round anchors of the nodes.

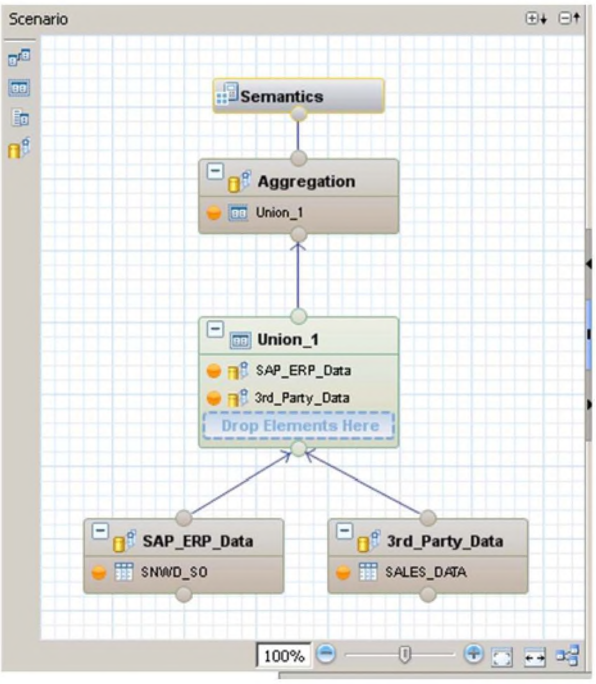


Figure 67: Connecting the Union_1 Node to the Aggregation node

2. In the *Aggregation* node, add the attributes *CLIENT* and *CURRENCY_CODE* to the output (non-aggregated columns), and the measures *GROSS_AMOUNT* and *NET_AMOUNT* as aggregated columns.

   a) Select the *Aggregation* node.

   b) In the *Details* pane, right-click the *CLIENT* column and choose *Add To Output*. Repeat this step for the *CURRENCY_CODE* column.

   c) Right-click the column *GROSS_AMOUNT* and choose *Add As Aggregated Column*. Repeat this step for the column *NET_AMOUNT*.

3. Save and activate the calculation view.

   a) Choose *Save and Activate.*

4. Preview the data of your Calculation View.

   a) Choose *Preview*.

   b) Select the *Raw Data* tab.

5. What is the value of GROSS_AMOUNT for CURRENCY_CODE: USD?
   a) *Answer:* 16 913.
6. Close the data preview.

> Note:
> An intermediate solution object is available at this stage:
>
> *STUDENT00 ha300 CVC_SO_00_STAGE1*

Task 7: Replace the Data Source for the 3rd_Party_Data Aggregation Node

You have just finished designing your calculation view, when you are informed that the third-party data source you took was not the right one. You must replace the *SALES_DATA* table with the *SALES_DATA_NEW* table in your model.

1. Open the calculation view *CVC_SO_##*, if needed.
2. In the Catalog, open the table *SALES_DATA_NEW* and compare the name of its columns with the columns of the table *SALES_DATA* that appear in the *3rd_Party_Data* node of the calculation view (in the *Details* pane).
   a) In the *Systems* view, expand the *Catalog* → *TRAINING* → *Tables* folder.
   b) Right-click the table *SALES_DATA_NEW* and choose *Open Definition*.
   c) Compare the column names with the names in the *Details* pane of the calculation view.
3. Replace the data source of the *3rd_Party_Data* node with the table *SALES_DATA_NEW* and to avoid any inconsistency in the calculation view, make sure that you correctly map the changed column names.

> Hint:
> Take care not to click *Finish* just after selecting the new data source; otherwise, the table will be replaced by the new one without any column mapping.
>
> Additionally, at the *Map Columns* step, note that you must first delete the inconsistent mapping by right-clicking the *AMOUNT* column in the target.

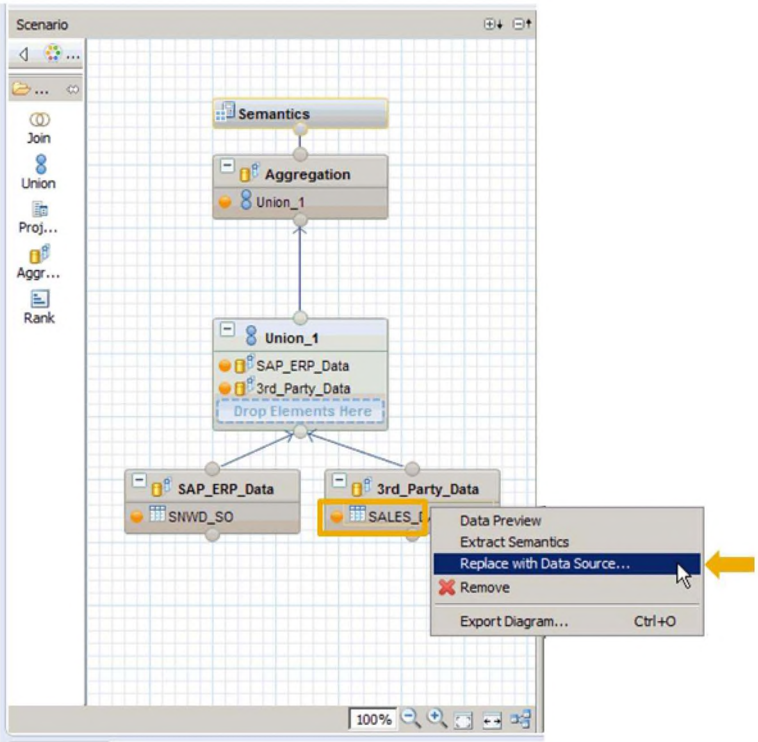


Figure 68: Replacing a Data Source

a) In the *3rd_Party_Data* node, right-click the *SALES_DATA* object and choose *Replace with Data Source*.

b) Enter **NEW** as the object name to search for, in the corresponding field, and select the *SALES_DATA_NEW (TRAINING)* object.

c) Choose *Next*.
The *Map Columns* step shows that the *AMOUNT* column in the target is mapped to a virtual *AMOUNT* column that does not exist in the new source.

d) To delete this non-relevant mapping, right-click the *AMOUNT* column in the target and choose *Remove Mapping*.

e) To define the correct mapping, drag the column *GROSS_AMOUNT* of the new source to the *AMOUNT* column of the previous target.

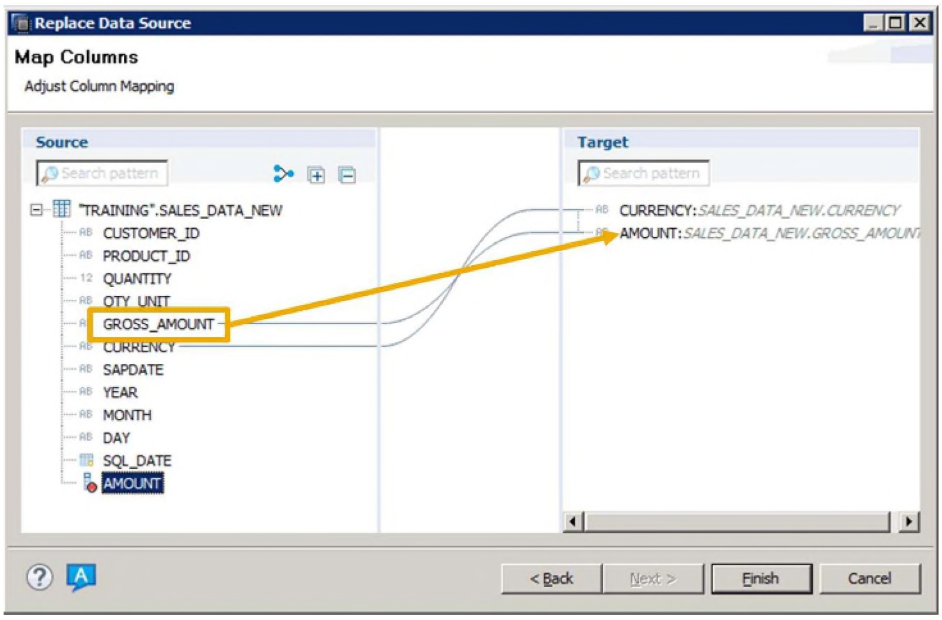


Figure 69: Mapping the GROSS_AMOUNT Column

f) Choose *Finish*.

4. Check the consistency of the Calculation view. In particular, in the output of the *3rd_Party_Data* node, check that the *AMOUNT* column has the correct mapping to the column *SALES_DATA_NEW.GROSS_AMOUNT*.

   a) Select the *3rd_Party_Data* node.

   b) In the *Output* pane, select the *AMOUNT* column.

5. Activate the Calculation View.

   a) Choose *Save and Activate*.

6. Preview the Data.

   a) Choose *Data Preview*.

   b) Select the *Raw Data* tab.

Aggregation Node

- In order to have further control on how the aggregation is calculated, it is beneficial to use the aggregation node in graphical calculation views.
- You can also add calculated columns with measures to the node. These calculations will be performed **after** aggregation.

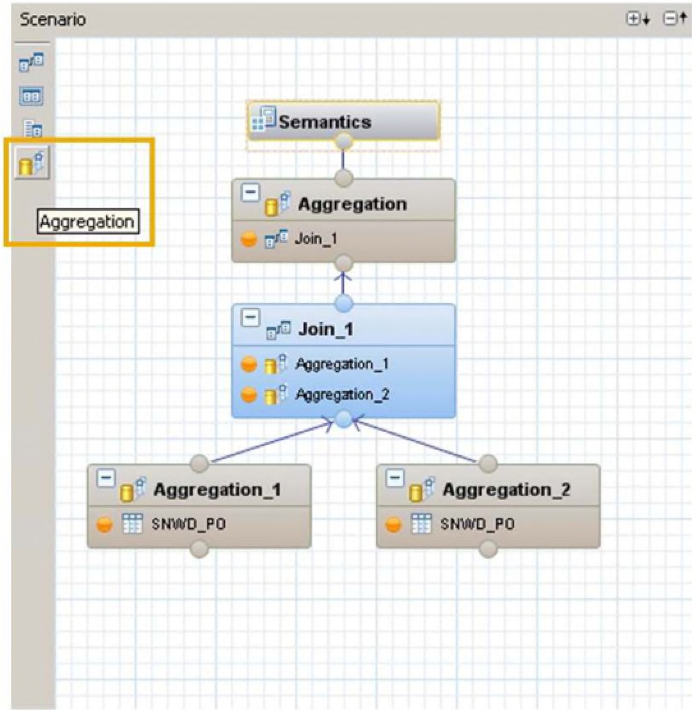


Figure 70: Aggregation Node

The purpose of an *Aggregation* node is to apply aggregate functions to measures based on one or several attributes.

> Note:
> This function acts similarly as a *GROUP BY* clause in a SQL query, where you must specify the aggregate function for each measure. Here, the *GROUP BY* clause is not materialized as such, but takes each and every attribute selected for the output.

A graphical Calculation View can support *SUM*, *MIN*, *MAX*, *COUNT*.

SAP HANA SPS11 allows you to apply the following additional aggregate functions to the Calculation Views::

- Average
- Variance
- Standard Deviation

> Caution:
> These aggregate functions should not be used in stacked scenarios. When you validate a model in such a scenario, a dedicated Validation Rule, *Validate Stacked Aggregations* generates a warning message.

Attribute vs. Aggregated Column

When adding a column to the output of an Aggregation node, you define whether the column is used for grouping (*Add to Output)* or if it must be combined with an aggregate function (*Add as Aggregated Column*). In the second case, you must specify the aggregate function to apply (*Sum* is applied by default).

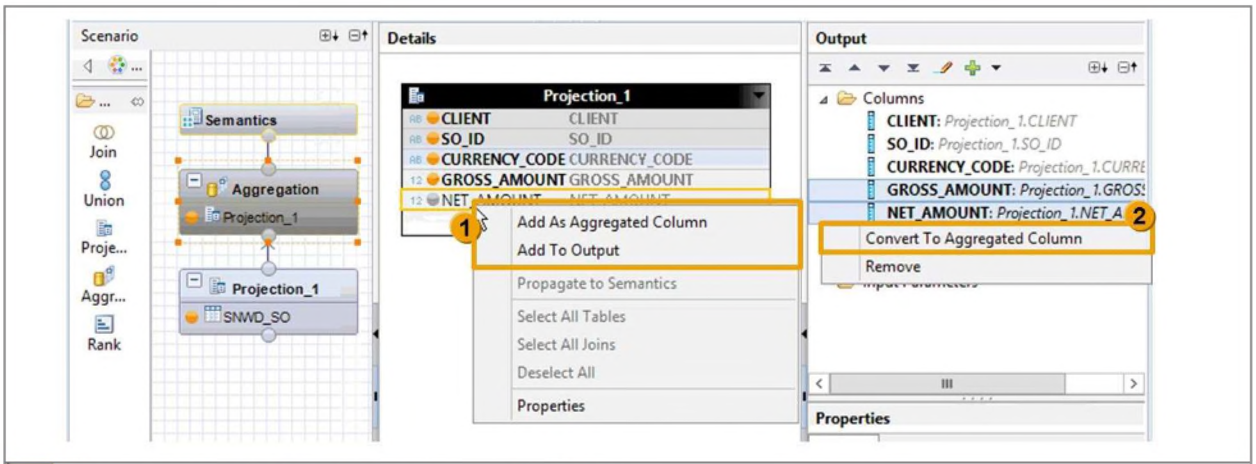


Figure 71: Adding Columns to an Aggregation Node

From SPS12 onwards, you can convert a column into an aggregated column in the *Output* pane, without losing the semantic information of the column, or the calculations defined on it in the upper nodes.

This feature is especially useful for calculation scenario design, for example when you are adding a new Aggregation node in the middle of a the calculation tree or when you convert a Projection node into an Aggregation node.

> Note:
> Before SPS12, you had to delete the column from the aggregation node, add it again, and propagate this column to all the upper nodes.

Controlling the Behavior of the Aggregation Node

When you work with Aggregation nodes, the list of columns requested by the client query can influence the way the aggregation is executed, especially in complex scenarios.

The following features can help you control the aggregation of measures, in order to build more flexible models:

- Keep Flag
- Transparent Filter

Keep Flag

Let's consider a scenario where a data source contains the details of sales orders, by Order ID. The only measures available are Quantity and Price.

> Note:
> Each Order ID relates to one Store, one Customer, and only one Product (in order to simplify the example).

You want to calculate the quantity and total sales for the product *Mouse* and the month of February.

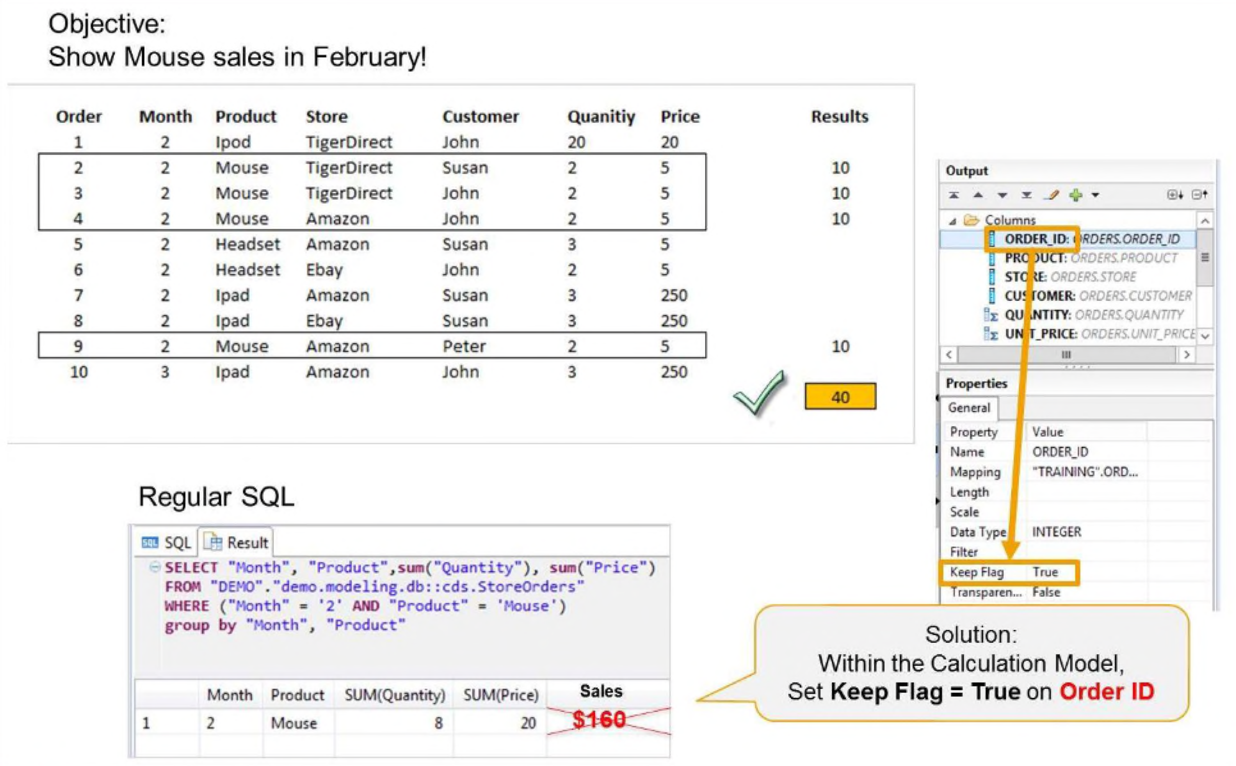


Figure 72: Keep Flag

Here, the issue is that the columns you request mandate a level of aggregation (Month, Product) that is generic. Hence, the Total Sales is calculated by multiplying the sum of quantities by the sum of unit prices.

Setting the *Keep Flag* property to true for the *Order ID* column forces the calculation to be triggered at the relevant level of granularity, even if this level is not the grouping level defined by the client query.

Transparent Filter

In some scenarios, using a filter (where clause) in a client query forces a column to be used in the *Group By* columns set.

In this scenario, for example, calculating the number of stores that sold mouses to John or Susan triggers an intermediate calculation of the storecount sum by Product and by Customer, which makes the total by Product irrelevant.

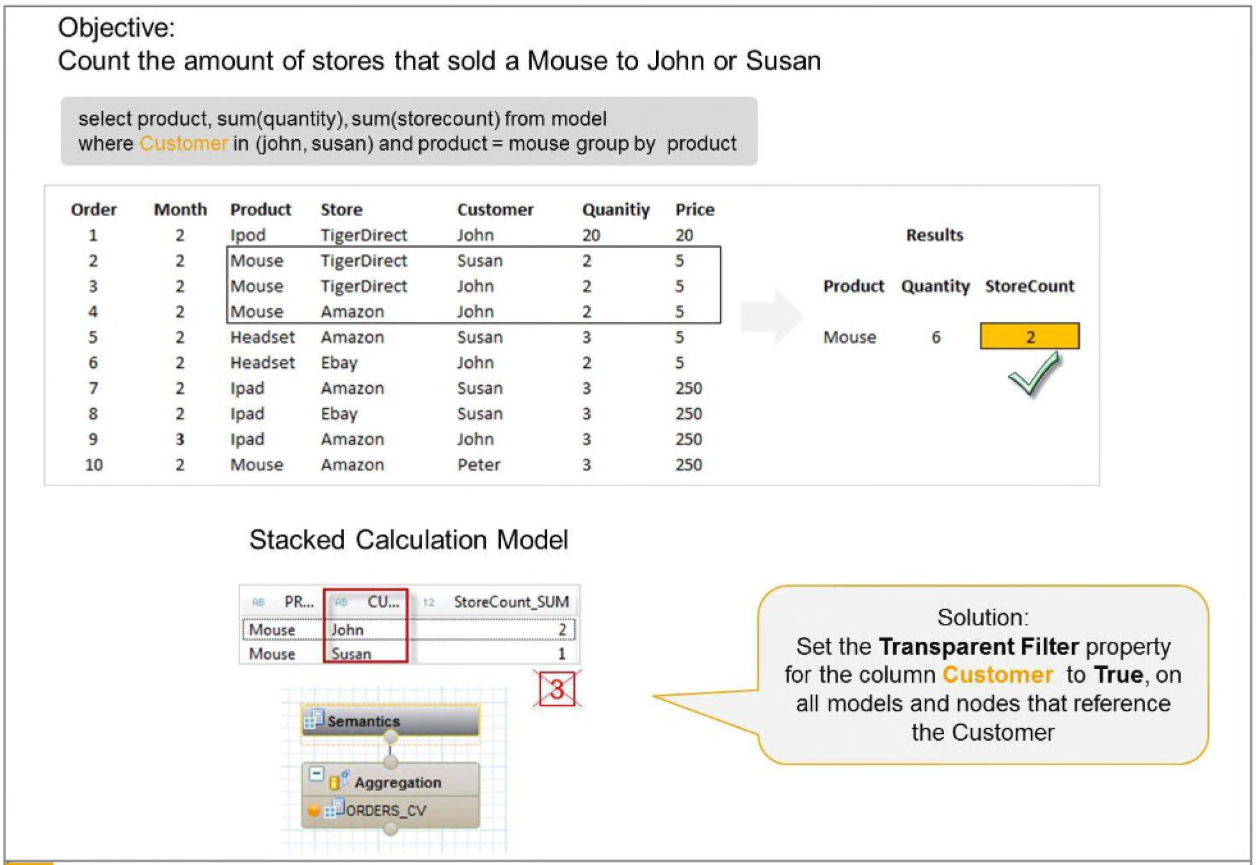


Figure 73: Transparent Filter

Setting the *Transparent Filter* property to true for all models and nodes that reference the *Customer* column, will stop the *Customer* column from being unnecessarily used in the *Group By* clause.

This flag is required in the following situations:

- When using stacked views where the lower views to have distinct count measures.
- When queries executed on the upper Calculation View contain filters on columns that are not projected.

Unit 1
Exercise 5

Control the Behavior of the Aggregation Node

Exercise Objectives

After completing this exercise, you will be able to:

- Control the behavior of the Aggregation node when you work with measures such as quantity and unit price.

Business Example

You want to analyze a set of data that represents sales orders for different products, stores and customers.

The source data includes the quantity and unit price for each sales order, but not the total price: you will have to set up your information view so that it computes correctly the total price regardless of what columns are requested by the client tool query.

In order to better understand how the view behaves, you will execute different SQL queries on top of the Calculation View in SAP HANA Studio.

> Note:
> For the sake of simplicity, there is only one product per sales order.

Task 1: Visualize the Source Data

> Note:
> Before attempting this task, open a file that contains several SQL statements which you will use during the exercise

1. In SAP HANA Studio, connected with your user *STUDENT##*, open the file *HA300 → Aggregation_Node_Column_Views_Queries.sql* and prepare the script by replacing ## with your group number everywhere.

   > Caution:
   > Until the end of this exercise, please make sure you execute only the SQL statements specified at each step. Do NOT execute the entire set of instructions from the SQL Console.

2. Execute the first query (under QUERY 1.a in your SQL Console) to visualize the content of the table that will be used during the exercise.

   The table is named *ORDERS* and located in the *TRAINING* schema.

```
SELECT * FROM "TRAINING"."ORDERS";
```

The table contains order details by order ID, product, shop, and customer. The quantity and unit price are in the table, but not the total price by line item (each order has only one line item).

Task 2: Create a Calculation View to Calculate the Total Sales

1. In your package *STUDENT##*, create a new Calculation View *CVC_ORDERS_AGR_##* with the following properties:

Property	Value
Name	**CVC_ORDERS_AGR_##** (## is your group number)
Description	**Sales Orders**
Type	*Graphical*
Data Category	*CUBE*
With Star Join	[Deselected]

2. Add a new Aggregation node *Aggregation_1* to the calculation view scenario, and define it as the data source for the *Aggregation* node.

Hint:
To align the nodes, at the bottom of the *Scenario* pane, choose *Auto Layout*.

3. Add the table *TRAINING.ORDERS* as the data source for the *Aggregation_1* node.

   Add all the columns from the table to the output, making sure that the columns *QUANTITY* and *UNIT_PRICE* are added as aggregate measures.

4. Propagate all the columns of the *Aggregation_1* node to the Semantics.

5. Create a calculated column **TOTAL_PRICE**, with the following properties:

Property	Value
Name	**TOTAL_PRICE**
Data Type	*Decimal*
Length	**15**
Scale	**2**
Expression	`"QUANTITY" * "UNIT_PRICE"`

   Then propagate this column to the semantics.

6. Assign the relevant type to the columns in the semantics:

   The *QUANTITY*, *UNIT_PRICE* and *TOTAL_PRICE* columns are *Measures*, other columns are *Attributes*.

7. Activate the Calculation View.

8. Preview the data with the Standard SAP HANA Functionality.
   The data preview seems consistent and the *TOTAL_PRICE* is correctly calculated.

Task 3: Query the New Calculation View

To investigate further the behavior of the Aggregation node, you will now execute several SQL statements on top of the Calculation View *CVC_ORDERS_AGR_##*. The main difference between the queries is the list of attribute columns and, in turn, the GROUP BY clause.

1. Execute a query to retrieve the quantities and price by Order ID and Product.

   The query is under QUERY 3.a in the SQL console that you opened at the beginning of the exercise.

```
SELECT
     "ORDER_ID",
     "PRODUCT",
     sum("QUANTITY") AS "QUANTITY",
     sum("UNIT_PRICE") AS "UNIT_PRICE",
     sum("TOTAL_PRICE") AS "TOTAL_PRICE"
FROM "_SYS_BIC"."STUDENT##/CVC_ORDERS_AGR_##"
GROUP BY "ORDER_ID",
     "PRODUCT";
```

2. Execute a query to retrieve the quantities and price only by Product.

   The query is under QUERY 3.b in the SQL console.

```
SELECT
     "PRODUCT",
     sum("QUANTITY") AS "QUANTITY",
     sum("UNIT_PRICE") AS "UNIT_PRICE",
     sum("TOTAL_PRICE") AS "TOTAL_PRICE"
FROM "_SYS_BIC"."STUDENT##/CVC_ORDERS_AGR_##"
GROUP BY
     "PRODUCT";
```

   What do you observe? How would you explain it?

> Note:
> Here, we keep the *UNIT_PRICE* column in the query to better understand how the total price has been computed. In real life, you would not sum up this column because this measure is not relevant when it is aggregated.

> Note:
> An intermediate solution object is available at this stage:
>
> *STUDENT00 → ha300 → CVC_ORDERS_AGR_00_STAGE1*

Task 4: Modify your Calculation View to Return Consistent Data

Modify your *CVC_ORDERS_AGR_##* Calculation View, to force the ORDER_ID column to participate in the *GROUP BY* clause of the view aggregation, even if it is not selected by the top query.

1. In the *Aggregation_1* node of your calculation view, set the *Keep Flag* property for the ORDER_ID column to *True*.

> Hint:
> Select the *ORDER_ID* column in the *Output* pane on the right of your screen.

2. Activate the Calculation View.
3. To retrieve the quantities and price only by Product, execute the previous SQL query again.

   The query is under QUERY 3.b in the SQL console.

```
SELECT
     "PRODUCT",
     sum("QUANTITY") AS "QUANTITY",
     sum("UNIT_PRICE") AS "UNIT_PRICE",
     sum("TOTAL_PRICE") AS "TOTAL_PRICE"
FROM "_SYS_BIC"."STUDENT##/CVC_ORDERS_AGR_##"
GROUP BY
     "PRODUCT";
```

   What do you observe?

4. To retrieve the quantities and total price by Product and Store, execute a query.

   The query is under QUERY 4.a in the SQL console.

```
SELECT
     "PRODUCT",
     "STORE",
     sum("QUANTITY") AS "QUANTITY",
     sum("TOTAL_PRICE") AS "TOTAL_PRICE"
FROM "_SYS_BIC"."STUDENT##/CVC_ORDERS_AGR_##"
GROUP BY
     "PRODUCT",
     "STORE";
```

   Is the result still consistent?

Unit 1
Solution 5

Control the Behavior of the Aggregation Node

Exercise Objectives

After completing this exercise, you will be able to:

- Control the behavior of the Aggregation node when you work with measures such as quantity and unit price.

Business Example

You want to analyze a set of data that represents sales orders for different products, stores and customers.

The source data includes the quantity and unit price for each sales order, but not the total price: you will have to set up your information view so that it computes correctly the total price regardless of what columns are requested by the client tool query.

In order to better understand how the view behaves, you will execute different SQL queries on top of the Calculation View in SAP HANA Studio.

> Note:
> For the sake of simplicity, there is only one product per sales order.

Task 1: Visualize the Source Data

> Note:
> Before attempting this task, open a file that contains several SQL statements which you will use during the exercise

1. In SAP HANA Studio, connected with your user *STUDENT##*, open the file *HA300 → Aggregation_Node_Column_Views_Queries.sql* and prepare the script by replacing ## with your group number everywhere.
   a) In the *Systems* view, right-click the entry *H00 (STUDENT##)* and choose *Open SQL Console*.
   b) Right-click the SQL Console and choose *Open File*.
   c) In the *Favorites* folder (top-left corner), expand the folder *HA300*.
   d) Double-click the file *Aggregation_Node_Column_Views_Queries.sql*.
   e) Choose *Edit → Find/Replace* or on your keyboard press Ctrl+F, fill in the *Find* and *Replace with* fields, and choose *Replace All*.

> Caution:
> Until the end of this exercise, please make sure you execute only the SQL statements specified at each step. Do NOT execute the entire set of instructions from the SQL Console.

2. Execute the first query (under QUERY 1.a in your SQL Console) to visualize the content of the table that will be used during the exercise.

   The table is named *ORDERS* and located in the *TRAINING* schema.

```
SELECT * FROM "TRAINING"."ORDERS";
```

   a) Highlight the specified statement and choose *Execute (F8)*.

   b) Review the output in the *Result* tab.

   The table contains order details by order ID, product, shop, and customer. The quantity and unit price are in the table, but not the total price by line item (each order has only one line item).

Task 2: Create a Calculation View to Calculate the Total Sales

1. In your package *STUDENT##*, create a new Calculation View *CVC_ORDERS_AGR_##* with the following properties:

Property	Value
Name	**CVC_ORDERS_AGR_##** (## is your group number)
Description	**Sales Orders**
Type	*Graphical*
Data Category	*CUBE*
With Star Join	[Deselected]

   a) In the content node of the *Systems* view, right-click your package and select *New → Calculation View*.

   b) Enter the details as shown in the table.

   c) Choose *Finish*.

2. Add a new Aggregation node *Aggregation_1* to the calculation view scenario, and define it as the data source for the *Aggregation* node.

> Hint:
> To align the nodes, at the bottom of the *Scenario* pane, choose *Auto Layout*.

   a) In the *Scenario* pane, drag and drop an Aggregation node from the palette to the area below the *Aggregation* Node.
   The new node is automatically named *Aggregation_1*.

b) Draw a connector between the top of the *Aggregation_1* node and the bottom of the *Aggregation* node.

c) To align the nodes, choose *Auto Layout* at the bottom of the *Scenario* pane.

3. Add the table *TRAINING.ORDERS* as the data source for the *Aggregation_1* node.

Add all the columns from the table to the output, making sure that the columns *QUANTITY* and *UNIT_PRICE* are added as aggregate measures.

a) In the *Scenario* pane, right-click the *Aggregation_1* node and choose *Add Objects*.

b) In the search criteria, enter **ORDERS**, select the *ORDERS (TRAINING)* table and choose *OK*.

c) In the *Details* pane, right-click the column *ORDER_ID* and choose *Add to Output*.

d) Repeat the last step for the other columns: *PRODUCT*, *STORE*, and *CUSTOMER*.

> Note:
> You can also select all the columns at once, on your keyboard press Ctrl while you select them.

e) Right-click the column *QUANTITY* and choose *Add as Aggregated Column*.

f) Repeat this step for the column *UNIT_PRICE*.

4. Propagate all the columns of the *Aggregation_1* node to the Semantics.

a) Select the *Aggregation_1* node, if needed.

b) In the *Output* pane, select all the columns (click the first one, then on your keyboard, press **Shift** and click the last one).

c) Right-click the selected columns and choose *Propagate to Semantics*.

5. Create a calculated column **TOTAL_PRICE**, with the following properties:

Property	Value
Name	**TOTAL_PRICE**
Data Type	*Decimal*
Length	**15**
Scale	**2**
Expression	`"QUANTITY" * "UNIT_PRICE"`

Then propagate this column to the semantics.

a) Select the *Aggregation_1* node, if needed.

b) In the *Output* pane, right-click the folder *Calculated Column* and choose *New*.

c) Enter the *Name*, *Data Type*, *Length* and *Scale* as specified.

d) Create the expression by using the *Elements* and *Operators* panes at the bottom of the screen.

e) Validate the expression syntax and choose *OK*.

f) Choose *OK*.

g) Right-click the new column and choose *Propagate to Semantics*.

6. Assign the relevant type to the columns in the semantics:

   The *QUANTITY*, *UNIT_PRICE* and *TOTAL_PRICE* columns are *Measures*, other columns are *Attributes*.

   a) Select the *Semantics* node.

   b) In the Details pane, choose *Auto Assign* at the right of the toolbar.

   c) Review the auto-assigned column types and, if needed, modify them.

7. Activate the Calculation View.

   a) Choose *Save and Activate*.

   b) In the *Job Log* view, check that the Activation job status is *Completed successfully*.

8. Preview the data with the Standard SAP HANA Functionality.

   a) Choose *Data Preview*.

   The data preview seems consistent and the *TOTAL_PRICE* is correctly calculated.

Task 3: Query the New Calculation View

To investigate further the behavior of the Aggregation node, you will now execute several SQL statements on top of the Calculation View *CVC_ORDERS_AGR_##*. The main difference between the queries is the list of attribute columns and, in turn, the GROUP BY clause.

1. Execute a query to retrieve the quantities and price by Order ID and Product.

   The query is under QUERY 3.a in the SQL console that you opened at the beginning of the exercise.

```
SELECT
     "ORDER_ID",
     "PRODUCT",
     sum("QUANTITY") AS "QUANTITY",
     sum("UNIT_PRICE") AS "UNIT_PRICE",
     sum("TOTAL_PRICE") AS "TOTAL_PRICE"
FROM "_SYS_BIC"."STUDENT##/CVC_ORDERS_AGR_##"
GROUP BY "ORDER_ID",
     "PRODUCT";
```

2. Execute a query to retrieve the quantities and price only by Product.

   The query is under QUERY 3.b in the SQL console.

```
SELECT
     "PRODUCT",
     sum("QUANTITY") AS "QUANTITY",
     sum("UNIT_PRICE") AS "UNIT_PRICE",
     sum("TOTAL_PRICE") AS "TOTAL_PRICE"
FROM "_SYS_BIC"."STUDENT##/CVC_ORDERS_AGR_##"
GROUP BY
     "PRODUCT";
```

   What do you observe? How would you explain it?

> Note:
> Here, we keep the *UNIT_PRICE* column in the query to better understand how the total price has been computed. In real life, you would not sum up this column because this measure is not relevant when it is aggregated.

a) Highlight the specified statement and choose *Execute (F8)*.

b) Review the output in the *Result* tab.
This time, the Total Price is not correctly calculated.

Because the Order ID attribute was not part of the Select statement, an inconsistent unit price, that is, the sum of the unit prices for each product, has been used to calculate the total sales by product.

> Note:
> An intermediate solution object is available at this stage:
>
> *STUDENT00 → ha300 → CVC_ORDERS_AGR_00_STAGE1*

Task 4: Modify your Calculation View to Return Consistent Data
Modify your *CVC_ORDERS_AGR_##* Calculation View, to force the ORDER_ID column to participate in the *GROUP BY* clause of the view aggregation, even if it is not selected by the top query.

1. In the *Aggregation_1* node of your calculation view, set the *Keep Flag* property for the ORDER_ID column to *True*.

> Hint:
> Select the *ORDER_ID* column in the *Output* pane on the right of your screen.

a) In the *Scenario* pane, select the *Aggregation_1* node.

b) In the *Output* pane, select the *ORDER_ID* column.

c) In the *Properties* pane, set the *Keep Flag* property to *True*.

2. Activate the Calculation View.

a) Choose *Save and Activate*.

3. To retrieve the quantities and price only by Product, execute the previous SQL query again.

The query is under QUERY 3.b in the SQL console.

```
SELECT
     "PRODUCT",
     sum("QUANTITY") AS "QUANTITY",
     sum("UNIT_PRICE") AS "UNIT_PRICE",
     sum("TOTAL_PRICE") AS "TOTAL_PRICE"
FROM "_SYS_BIC"."STUDENT##/CVC_ORDERS_AGR_##"
GROUP BY
     "PRODUCT";
```

What do you observe?

a) Highlight the specified statement and choose *Execute (F8)*.

b) Review the output in the *Result* tab.
The Total Price has been correctly calculated, even with summarized data by Product. The *Keep Flag* property has enforced the computation of the Total Price at the Order ID level, where this measure can be calculated as Quantity * Unit Price. Any further aggregation (by product, store or customer) will return consistent results.

4. To retrieve the quantities and total price by Product and Store, execute a query.

The query is under QUERY 4.a in the SQL console.

```
SELECT
     "PRODUCT",
     "STORE",
     sum("QUANTITY") AS "QUANTITY",
     sum("TOTAL_PRICE") AS "TOTAL_PRICE"
FROM "_SYS_BIC"."STUDENT##/CVC_ORDERS_AGR_##"
GROUP BY
     "PRODUCT",
     "STORE";
```

Is the result still consistent?

a) Highlight the specified statement and choose *Execute (F8)*.

b) Review the output in the *Result* tab.
The result from the query is still consistent: you can group the result by different attributes in the query you run on top of the Calculation View.

Rank Node

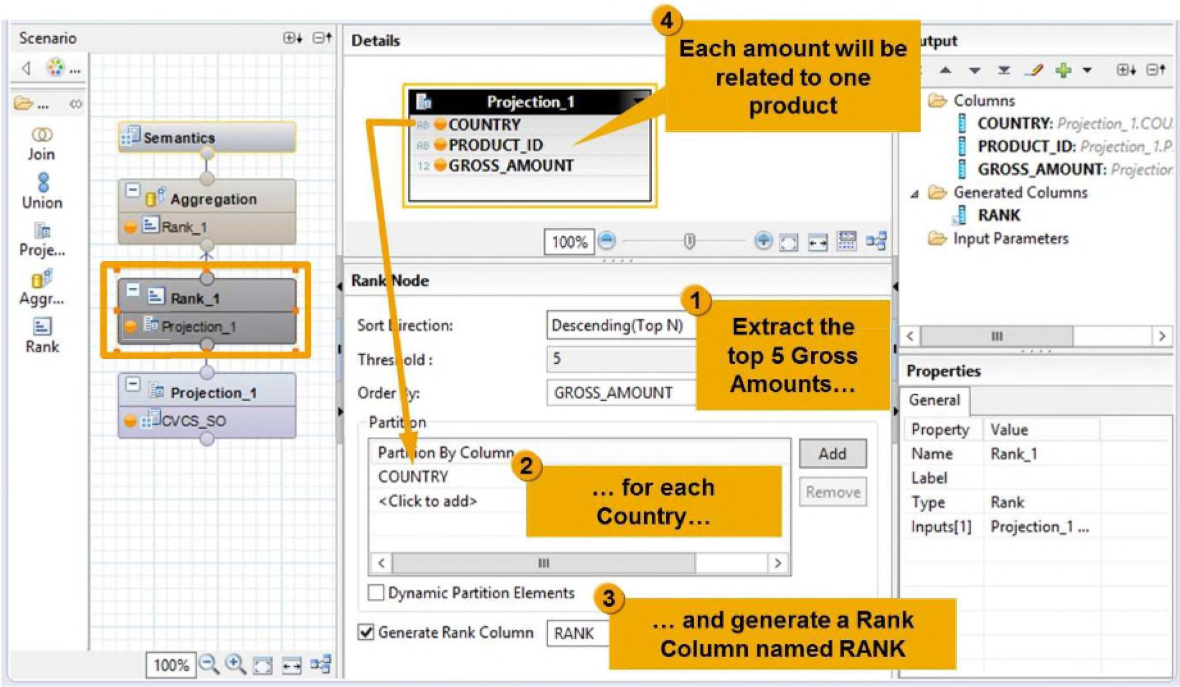


Figure 74: Rank Node

The purpose of the *Rank* node is to enable the selection, within a data set, of the Top or Bottom 1, 2, … n values for a defined measure, and to output these measures together with the corresponding attributes and, if needed, other measures.

For example, with a *Rank* node, you can easily build a subset of data that gives you, the five products that generated the biggest revenues (considering the measure *Gross Amount*), for each country.

Main Settings of a Rank Node

The main settings of a *Rank* node are as follows:

Table 4: Main Settings of a Rank Node

Setting	Purpose
Sort Direction	Descending (Top N) or Ascending (Bottom N).
Threshold	Number of rows returned for each Partition (N).
ORDER BY	The column that is used to order the data and extract the top or bottom N values.
Partition	One or several columns by which you want to provide the top or bottom N values. The result will give you N values for each partition "node".
Dynamic Partition Elements	Define whether the partition can be adjusted automatically based on the columns that are selected by an upper node, an upper view, or query you run on top of the view.
Generate Rank Column	Whether you want to output a column with rank position (with values 1 to N).

The top or bottom N values that are returned by the rank node depend totally on the attributes that you use in the node, project with an upper node, or select with the query you run on top of the view.

Considering the same example (for each Region, select the Top 5 Products in terms of Gross Revenue), let's describe how the Rank node will behave if you modify the list of attributes in the output:

- If you remove the *PRODUCT_ID* attribute from the output:

  The Rank node no longer behaves as a rank. It just sums up Gross Amount by country, and each row in the output has the rank 1.

- You add an attribute *BP_COMPANY_NAME* to the output:

  Each amount is now related to a Product and Business Partner pair. That is, for each Country, you get the 5 biggest revenues generated by Product and Business Partner in each country.

Assigning a Type to the Rank Column

Depending on how you want to use the ranking information, you can decide to assign to the rank column the type Attribute or Measure.

Rank Column: Measure or Attribute?

- Attribute

  Assigning the type *Attribute* is a simple approach.

  It is probably a bit less error-prone, because you are not tempted to perform an irrelevant aggregation of ranking positions. Removing the rank column from the top query will just apply the defined aggregate function, very often SUM, to the ranked values of each partition.

- Measure

  With the type *Measure*, the rank column provides more flexibility.

  For example, if you set the default aggregate function to MAX, you can retrieve summarized data, such as the total sales generated by the five biggest orders in each country, while keeping the information about how many orders are actually totalled in each country. Indeed, there might be countries that have received less than five orders over the considered period, and this information could be of interest when analyzing the data.

Unit 1
Exercise 6

Extract Top N Values with a Calculation View

Exercise Objectives

After completing this exercise, you will be able to:

- Extract the Top-N values from a data set, by using a Rank Node.
- Understand the impact of the client tool query on the way data are retrieved.
- Define the rank node so that it adapts dynamically to the selected columns.

Business Example

Your company wants to analyze its sales orders by extracting the top 5 products ordered (based on the Gross Amount) in each country.

You will first create a new Calculation View with a rank node that fulfills exactly this requirement, based on a provided Calculation View retrieving the Sales Order Items.

You will explore the capabilities of the Rank node that offer more dynamic handling for columns.

Task 1: Create a new Calculation View Using a Rank Node
Create a Calculation View that will process a Top–5 extraction from a provided Sales Order Items View.

1. Create a new Calculation View of type *CUBE*, with the following properties:

Field	Value
Name	**CVC_SO_RANK_##** (## is your group number)
Description	**Sales Orders Rank**
View Type	*Calculation View*
Subtype	*Graphical*
Data Category	*CUBE*
With Star Join	[Deselected]

2. Add a new Rank node *Rank_1* to the calculation view scenario, and define it as the data source for the *Aggregation* node.

3. Define the view *training* → *ha300* → *CVCS_SO* as the data source for the *Rank_1* node, and add the following columns to the output:

   - COUNTRY
   - PRODUCT_ID

- GROSS_AMOUNT

> Hint:
> You can also add the columns directly to the output of the projection node by using the command *Propagate to Semantics* from the context menu of the *Rank_1* source.

4. Define the settings of the Rank node so that it performs the following data extraction:

   - Extract, for each country, the 5 products that total the biggest sales orders amount (column: *GROSS_AMOUNT*).
   - A rank column, named *RANK*, is generated.

> Caution:
> Do not forget to propagate this new column up to the *Semantics* node.

5. Activate the *CVC_SO_RANK_##* Calculation View.

> Note:
> From now on, any SQL statement mentioned in this exercise can be found in a specific file in your *student* folder:
> *HA300 → Rank_Node_Column_Views_Queries.sql*.

6. In a new SQL Console, connected with your user *STUDENT##*, open the file *HA300 → Rank_Node_Column_Views_Queries.sql* and prepare the script by replacing ## with your group number everywhere.

> Caution:
> Until the end of this exercise, please make sure you execute only the SQL statements specified at each step. Do NOT execute the entire set of instructions from the SQL Console.

7. Preview the data from the new Calculation View *CVC_SO_RANK_##*.

   To better control how the data are ordered, instead of using the *Data Preview* feature of SAP HANA Studio, open a SQL Console and execute the following statement (under QUERY 1.a in your SQL console).

```
SELECT
     "COUNTRY",
     "PRODUCT_ID",
     sum("GROSS_AMOUNT") AS "GROSS_AMOUNT",
     sum("RANK") AS "RANK"
FROM "_SYS_BIC"."STUDENT##/CVC_SO_RANK_##"
GROUP BY "COUNTRY",
     "PRODUCT_ID"
ORDER BY "COUNTRY",
     "RANK"
```

> Hint:
> From the *Systems* view, you can right-click the view name *CVC_SO_RANK_##* and choose *Generate Select SQL*. Then add the *ORDER BY* clause at the end of the statement.

8. Compare with the data computed from the source Calculation View *training* → *ha300* → *CVCS_SO*, for the country *US*.

   In the SQL Console, execute the following statement (under QUERY 1.b in your SQL console):

```
SELECT
     "COUNTRY",
     "PRODUCT_ID",
     sum("GROSS_AMOUNT") AS "GROSS_AMOUNT"
FROM "_SYS_BIC"."training.ha300/CVCS_SO"
WHERE "COUNTRY"='US'
GROUP BY "COUNTRY",
     "PRODUCT_ID"
ORDER BY "COUNTRY",
     "GROSS_AMOUNT" DESC
```

> Note:
> An intermediate solution object is available at this stage:
>
> *STUDENT00* → *ha300* → *CVC_SO_RANK_00_STAGE1*

Task 2: Modify the Calculation View to Better Control Rank Behavior

Modify the properties of the Rank node and Output columns of the Calculation View *CVC_SO_RANK_##*. You will test the view behavior by executing several queries on top of the Calculation View.

1. Add the column *BP_COMPANY_NAME* to the output of the *Rank_1* node and up to the Semantics node.

2. Activate the Calculation View.

3. Execute again the first SQL query against the view. This query is under QUERY 1.a in your SQL Console.

   What do you notice?

4. Execute a similar query, but this time include the BP_COMPANY_NAME column.

   The query is under QUERY 2.a in your SQL console:

```
SELECT
     "COUNTRY",
     "BP_COMPANY_NAME",
     "PRODUCT_ID",
     sum("GROSS_AMOUNT") AS "GROSS_AMOUNT",
     sum("RANK") AS "RANK"
FROM "_SYS_BIC"."STUDENT##/CVC_SO_RANK_##"
GROUP BY "COUNTRY",
     "BP_COMPANY_NAME",
     "PRODUCT_ID"
ORDER BY "COUNTRY",
     "RANK"
```


What do you notice this time?

5. Add the *BP_COMPANY_NAME* column to the *Partition by Column* list of the Rank node and activate the Calculation View.
6. Execute again the first SQL query against the view. This query is under QUERY 1.a in your SQL Console.

   What do you notice?
7. Activate the *Dynamic Partition Elements* in the properties of the *Rank_1* node and activate the Calculation View.
8. Execute the first SQL query against the view again. This query is under QUERY 1.a in your SQL Console.

   What do you notice?

Unit 1
Solution 6

Extract Top N Values with a Calculation View

Exercise Objectives

After completing this exercise, you will be able to:

- Extract the Top-N values from a data set, by using a Rank Node.
- Understand the impact of the client tool query on the way data are retrieved.
- Define the rank node so that it adapts dynamically to the selected columns.

Business Example

Your company wants to analyze its sales orders by extracting the top 5 products ordered (based on the Gross Amount) in each country.

You will first create a new Calculation View with a rank node that fulfills exactly this requirement, based on a provided Calculation View retrieving the Sales Order Items.

You will explore the capabilities of the Rank node that offer more dynamic handling for columns.

Task 1: Create a new Calculation View Using a Rank Node

Create a Calculation View that will process a Top–5 extraction from a provided Sales Order Items View.

1. Create a new Calculation View of type *CUBE*, with the following properties:

Field	Value
Name	**CVC_SO_RANK_##** (## is your group number)
Description	**Sales Orders Rank**
View Type	*Calculation View*
Subtype	*Graphical*
Data Category	*CUBE*
With Star Join	[Deselected]

   a) In the content node of the *Systems* view, right-click on your package *STUDENT##* and select *New → Calculation View*.

   b) Enter the details from the table.

   c) Choose *Finish*.

2. Add a new Rank node *Rank_1* to the calculation view scenario, and define it as the data source for the *Aggregation* node.

a) In the *Scenario* pane, drag and drop a Rank node from the palette to the area below the Aggregation node.
The new node is automatically named Rank_1.

b) Draw a connector between the top of the *Rank_1* node and the bottom of the *Aggregation* node.

3. Define the view *training* → *ha300* → *CVCS_SO* as the data source for the *Rank_1* node, and add the following columns to the output:

- COUNTRY
- PRODUCT_ID
- GROSS_AMOUNT

> Hint:
> You can also add the columns directly to the output of the projection node by using the command *Propagate to Semantics* from the context menu of the *Rank_1* source.

a) In the *Scenario* pane, right-click the *Rank_1* and choose *Add Objects*.

b) In the search field, enter **cvcs_so**. Then select *the CVCS_SO (training.ha300)* view and click *OK*.

c) In the *Details* pane, right-click the *COUNTRY* column and choose *Propagate to Semantics*.

d) Repeat the previous step for the two other columns.

e) Check that the three columns are added to the *Semantics* node, with the relevant type (Attribute or Measure).

4. Define the settings of the Rank node so that it performs the following data extraction:

- Extract, for each country, the 5 products that total the biggest sales orders amount (column: *GROSS_AMOUNT*).
- A rank column, named *RANK*, is generated.

> Caution:
> Do not forget to propagate this new column up to the *Semantics* node.

a) In the *Rank Node* pane, define the rank properties as follows:

Property	Value
Sort direction	*Descending* (default value)
Threshold	**5**
Order by	*GROSS_AMOUNT*
Partition by column	*COUNTRY*

Property	Value
Generate rank column	[Selected]
Generated rank column name	*RANK*

b) In the *Output* pane, right-click the Generated Column *RANK* and choose *Propagate to Semantics*.

c) In the *Propagate to Semantics* dialog box, click *OK*.

5. Activate the *CVC_SO_RANK_##* Calculation View.

   a) Choose *Save and Activate*.

> Note:
> From now on, any SQL statement mentioned in this exercise can be found in a specific file in your *student* folder:
> *HA300 → Rank_Node_Column_Views_Queries.sql*.

6. In a new SQL Console, connected with your user *STUDENT##*, open the file *HA300 → Rank_Node_Column_Views_Queries.sql* and prepare the script by replacing ## with your group number everywhere.

   a) In the *Systems* view, right-click the entry *H00 (STUDENT##)* and choose *Open SQL Console*.

   b) Right-click the SQL Console and choose *Open File*.

   c) In the *Favorites* folder (top-left corner), expand the folder *HA300*.

   d) Double-click the file *Rank_Node_Column_Views_Queries.sql*.

   e) Choose *Edit → Find/Replace* or press **`Ctrl+F`**, fill in the *Find* and *Replace with* fields, and choose *Replace All*.

> Caution:
> Until the end of this exercise, please make sure you execute only the SQL statements specified at each step. Do NOT execute the entire set of instructions from the SQL Console.

7. Preview the data from the new Calculation View *CVC_SO_RANK_##*.

   To better control how the data are ordered, instead of using the *Data Preview* feature of SAP HANA Studio, open a SQL Console and execute the following statement (under QUERY 1.a in your SQL console).

```
SELECT
     "COUNTRY",
     "PRODUCT_ID",
     sum("GROSS_AMOUNT") AS "GROSS_AMOUNT",
     sum("RANK") AS "RANK"
FROM "_SYS_BIC"."STUDENT##/CVC_SO_RANK_##"
GROUP BY "COUNTRY",
     "PRODUCT_ID"
```

```
ORDER BY "COUNTRY",
     "RANK"
```

> Hint:
> From the *Systems* view, you can right-click the view name *CVC_SO_RANK_##* and choose *Generate Select SQL*. Then add the *ORDER BY* clause at the end of the statement.

a) Highlight the specified statement and choose *Execute (F8)*.

b) Alternatively, generate the Select SQL statement from the context menu of the view, add the *ORDER BY* clause and choose *Execute (F8)*.

8. Compare with the data computed from the source Calculation View *training → ha300 → CVCS_SO*, for the country *US*.

   In the SQL Console, execute the following statement (under QUERY 1.b in your SQL console):

```
SELECT
     "COUNTRY",
     "PRODUCT_ID",
     sum("GROSS_AMOUNT") AS "GROSS_AMOUNT"
FROM "_SYS_BIC"."training.ha300/CVCS_SO"
WHERE "COUNTRY"='US'
GROUP BY "COUNTRY",
     "PRODUCT_ID"
ORDER BY "COUNTRY",
     "GROSS_AMOUNT" DESC
```

a) Highlight the specified statement and choose *Execute (F8)*.

b) Alternatively, generate a Select SQL statement from the context menu of the view *CVCS_SO*, modify it according to the code provided above, and choose *Execute (F8)*.

> Note:
> An intermediate solution object is available at this stage:
>
> *STUDENT00 → ha300 → CVC_SO_RANK_00_STAGE1*

Task 2: Modify the Calculation View to Better Control Rank Behavior

Modify the properties of the Rank node and Output columns of the Calculation View *CVC_SO_RANK_##*. You will test the view behavior by executing several queries on top of the Calculation View.

1. Add the column *BP_COMPANY_NAME* to the output of the *Rank_1* node and up to the Semantics node.

   a) In the Details of the **Rank_1** node, right-click the column *BP_COMPANY_NAME* and choose *Propagate to Semantics*.

2. Activate the Calculation View.

   a) Choose *Save and Activate*.

3. Execute again the first SQL query against the view. This query is under QUERY 1.a in your SQL Console.

   What do you notice?

a) Highlight the specified statement and choose *Execute (F8)*.
   The result is not impacted by the fact adding a new column if this column is not listed in the query that consumes the Calculation View.

4. Execute a similar query, but this time include the BP_COMPANY_NAME column.

   The query is under QUERY 2.a in your SQL console:

```
SELECT
     "COUNTRY",
     "BP_COMPANY_NAME",
     "PRODUCT_ID",
     sum("GROSS_AMOUNT") AS "GROSS_AMOUNT",
     sum("RANK") AS "RANK"
FROM "_SYS_BIC"."STUDENT##/CVC_SO_RANK_##"
GROUP BY "COUNTRY",
     "BP_COMPANY_NAME",
     "PRODUCT_ID"
ORDER BY "COUNTRY",
     "RANK"
```

   What do you notice this time?

   a) Highlight the specified statement and choose *Execute (F8)*.
      There are still five top values generated by country, but based on the total sales amount by customer/product pair.

      The columns that are not specified in the *Partition by Column* list of the Rank node determine at which level the measure is analyzed.

5. Add the *BP_COMPANY_NAME* column to the *Partition by Column* list of the Rank node and activate the Calculation View.

   a) In the *Scenario* pane, select the *Rank_1* node.

   b) In the *Rank Node* pane, under the *Partition* area, click *<Click to add>*.

   c) Double-click the *BP_COMPANY_NAME* column.

6. Execute again the first SQL query against the view. This query is under QUERY 1.a in your SQL Console.

   What do you notice?

   a) Highlight the specified statement and choose *Execute (F8)*
      The results are not consistent, because the column *BP_COMPANY_NAME* is included in the *Partition by Column* list but not part of the top query applied to the view

7. Activate the *Dynamic Partition Elements* in the properties of the *Rank_1* node and activate the Calculation View.

   a) In the *Scenario* pane, select the *Rank_1* node.

   b) In the *Rank Node* pane, select the *Dynamic Partition Elements* checkbox.

   c) Click *Save and Activate*.

8. Execute the first SQL query against the view again. This query is under QUERY 1.a in your SQL Console.

   What do you notice?

   a) Highlight the specified statement and choose *Execute (F8)*.
      The results are consistent, even if the column *BP_COMPANY_NAME* is not part of the top query applied to the view.

The column is ignored when it is not part of the top SQL statement.

b) If the column *BP_COMPANY_NAME* were part of the top query, the outcome would be a rank by country and company pair.

You can validate this final result by executing the following query (under QUERY 2.b in your SQL console):

```
SELECT
     "COUNTRY",
     "BP_COMPANY_NAME",
     "PRODUCT_ID",
     sum("GROSS_AMOUNT") AS "GROSS_AMOUNT",
     sum("RANK") AS "RANK"
FROM "_SYS_BIC"."STUDENT##/CVC_SO_RANK_##"
GROUP BY "COUNTRY",
     "BP_COMPANY_NAME",
     "PRODUCT_ID"
ORDER BY "COUNTRY",
     "BP_COMPANY_NAME",
     "RANK"
```

Specific Features to Enhance Flexibility of Calculation Views Design

Because Calculation Views offer a lot of flexibility, in particular the possibility to have a large number of nodes in a view, SAP HANA Studio provides advanced functionality to manage nodes and semantics.

Feature	Purpose
Switching Node Types	Change a node type with another one without losing the link to the data source/lower and upper nodes
Replacing a Data Source	Replace a node source by another one without losing the output columns (throughout all the upper nodes), calculated columns, etc.
Insert/Remove a Node Within the Calculation Tree	Insert an additional calculation node (or remove an existing one) without losing the link between columns end-to-end.
Extract Semantics	Apply to the Semantics of the Calculation view the semantics from an underlying node or data source.
Propagate Semantics	Transfer the semantics of a column from a view node to the semantics.
Previewing the Output of Intermediate Nodes	To troubleshoot problems in view design, preview the data of an intermediate node rather than the whole Calculation View.
Map Input Parameters between Nodes	Map Input Parameters of a view to input parameters defined in underlying (source) views of other nodes.

Figure 75: Useful Features for Calculation View Definition

Switching Between Certain Types of Node

It is possible to convert a *Projection* node into an *Aggregation* node and vice versa without losing the reference to the data sources of the node.

Additionally, in the particular case of the upper node of a calculation view (that is, the last node of the calculation logic), you can switch between *Projection*, *Aggregation*, and *Star Join* node.

Replacing the Data Sources of a Node

Depending of the type of node and view configuration, you can replace the data source of a node (for example, a table, an information view, or another node) by another one.

This functionality is useful, for example, when you want to have the columns still propagated to all the upper nodes. You can also keep the existing calculated columns defined in this node working despite the change of data source.

When you do the change, a column mapping dialog box appears. It helps you assign the columns of the new data source to the output columns previously defined in the node.

Insert or Remove a Node Within the Calculation Tree

It is possible to insert an additional node (except the *Star Join* one) between two existing nodes, without loosing the selection of columns for the output, and all the related information (calculated columns, and so on).

Hint:
To do this task, you must drag the required node icon from the Palette to the connector where you want to create the additional node.

You can also insert a new node between a node and its data source.

Hint:
Drag the required node icon from the Palette to the data source until the mouse pointer shows a + sign.

Similarly, you can remove an intermediate node from a calculation tree, and link together the next and previous nodes.

Hint:
In that case, do not use the *Delete* command. Instead, do the following:

- If you do not want to use the replaced node later on, right-click the node and choose *Remove and replace with* → *Node*. In the *Replace with* view, choose the node you want to use as a replacement, define the relevant column mapping and choose *Finish*.
- If you want to keep the node for later use, right-click the name of the node that appears in the next node, and choose *Replace with* → *Node*. In the *Replace with* view, choose the node you want to use as a replacement, define the relevant column mapping and choose *Finish*.

Extracting Semantics

It is possible to extract the semantics from the underlying data sources of the nodes (especially when these data sources are information views with rich semantics) and to propagate them to the semantics of the calculation view.

Previewing the Output of Intermediate Nodes

To understand and fine-tune the different steps of a calculation view, each node can be previewed independently from the Calculation view itself.

To achieve this preview, the view must be active, that is, it must have been successfully validated and activated.

Note:
After the Calculation View has been activated and modified, you have the ability to execute the data preview of a node that was not changed, again (which means, neither the node itself or the lower nodes have been modified). However, you receive a warning that it might give a wrong data result.

Map Input Parameters between Nodes

You can map Input Parameters (and also variables) defined in a calculation view with the Input Parameters/variables defined in the source views. You can do that to pass the parameter value itself, but also to pass the relevant list of values to the input parameter dialog box at run-time (when the view is executed).

> Note:
> In the *Manage Mappings* view, you can even copy and map an input parameter from a source view.
>
> That is, you do not need to create a new input parameter in the Calculation View (and define all its settings) before mapping it. This task can be done automatically in just one operation.

Copy and Paste Part of a Scenario

From SAP HANA SPS10, it is now possible to copy and paste part of a model in the scenario pane within the same Calculation View.

Caching View Results

In SAP HANA, it is possible to cache the results of a Calculation View, in order to speed up the time needed to get the results when executing a query against the view. This feature is useful when you have complex scenarios or big amounts of data.

Caching view data involves two types of settings, at the SAP HANA database level, and at the individual view level.

General Settings for View Results Cache

The possibility to cache query results must be turned on by the SAP HANA database administrator.

The system administrator defines some key options, such as the minimum duration of a query that triggers caching for a view. Indeed, if the query results are extremely fast to generate, it might not be useful to use caching.

In addition, because caching view data can use a lot of space on the database system, it is possible to define a size limit for the cache.

Specific Cache Settings at the View Level

At the view level, you must explicitly require results caching in the Calculation View properties.

You have to select a mode for cache invalidation:

- Hourly

  The cache is no longer valid after one hour

- Daily

  The cache is no longer valid after one day

- Transactional

  The cache is invalidated whenever a transaction affects one of the source tables for the information view.

Data Lineage

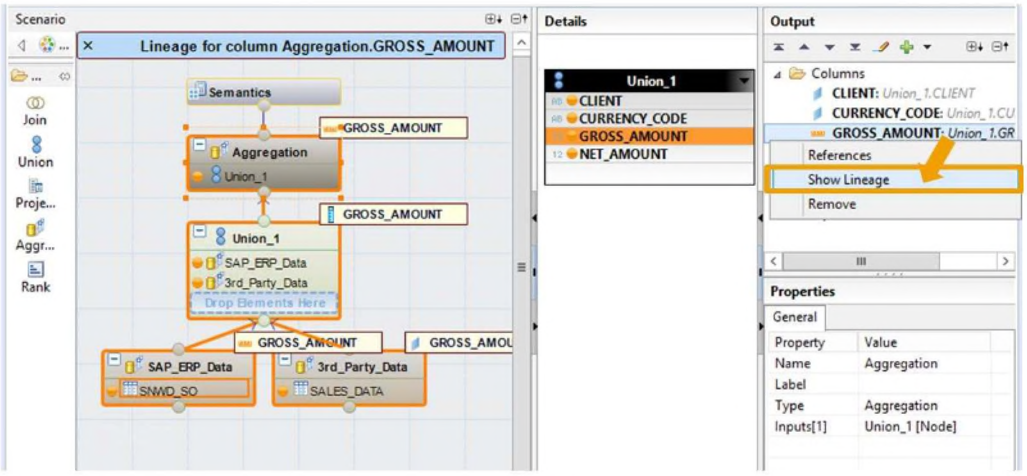


Figure 76: Data Lineage

The data lineage functionality enables you to track the origin of a column along the calculation scenario, and down to the first node where it appears. This node can be a bottom-level node of the view where the column is selected from a source, such as another information view, a column table, and so on. It can also be an intermediate node where the column is calculated.

Data lineage is a great feature for locating the origin of columns that might have been renamed during the flow.

LESSON SUMMARY

You should now be able to:

- Use measures in Calculation Views
- Explain the benefits of each type of node in Calculation Views
- Create and combine nodes in Calculation Views
- Use the features of Calculation Views to enhance the flexibility of this type of view

Unit 1
Lesson 5

Understanding Deprecated Graphical Views

LESSON OVERVIEW

As part of the recent evolution of the modeling approach for graphical information views, SAP now recommends that you only use Calculation Views.

This lesson will give you the essential knowledge about the specifics of Attribute and Analytic Views, because you might have seen them in previous implementations of SAP HANA.

Business Example

You work on an SAP HANA project for a customer who has decided to upgrade their SAP HANA Systems to SAP HANA SPS11.

You are in charge of analyzing the current set of graphical information models, and to prepare the transition to the new modeling approach with Calculation Views only.

For this purpose, you must know more about Attribute and Analytic Views, and in particular check whether the customer used specific features of this type of view that are not yet supported by Calculation Views.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Explain Attribute Views and Analytic Views (Deprecated View Types)

Deprecated Graphical Calculation Views

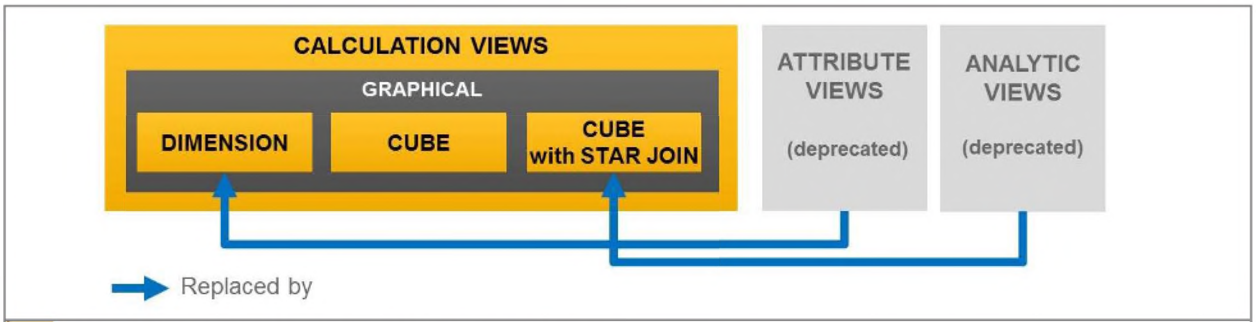


Figure 77: Deprecated Graphical Calculation Views

SAP HANA releases have brought a lot of enhancements to the functional coverage and performance of graphical Calculation Views. As a result, the use of Attribute and Analytic Views is no longer recommended.

The optimization of Calculation Views allows the distribution of data processing to the most relevant SAP HANA engine (Join engine, OLAP engine, and Calculation engine).

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

> Note:
> In particular, the optimizer will always evaluate the possibility to delegate data processing for *Cube with Star Join* Calculation Views to the same operators as the ones used by Analytic Views.

Features from Deprecated Views Not Supported in Calculation Views

As part of the shift in the modeling approach, you should be aware of a few use cases where Calculation Views (up to SAP HANA SPS12) does not provide the same functionality as Attribute and Analytic Views.

Table 5: Functional Difference between Information Views Types

Feature	Available only in	Comments
Fuzzy Search	Attribute Views	It is possible to enable one or several columns of Attribute Views for a Fuzzy Search. This is not possible in Calculation Views.
Derived Attribute View	Attribute Views	If an Attribute View must be used several times in the same Analytic View, you can create a Derived Attribute View. This is a dynamic copy of the base Attribute View.

Apart from these limitations, Calculation Views are definitely the recommended graphical modeling object type.

Derived Attribute Views

In some business cases, it is required to use the same Attribute View more than once in the Star Join of a single Analytic View. In such cases, it is possible to derive an Attribute View, where the derived view just acts as a reference to the base Attribute View.

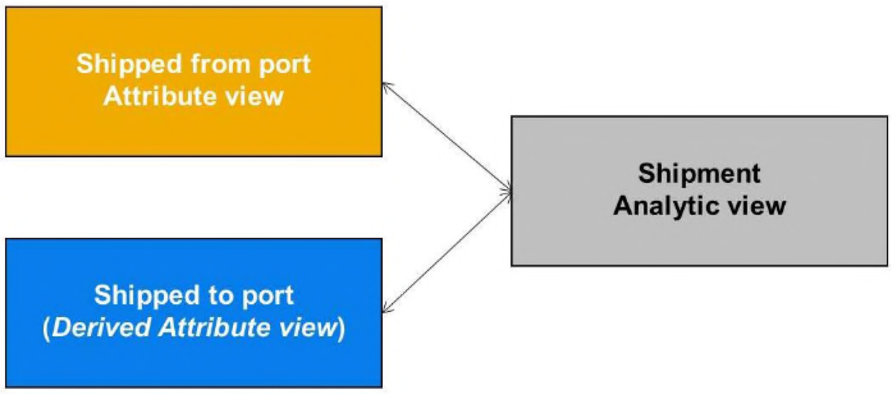


Figure 78: Derived Attribute View – Example

In this example, only one Attribute View is needed for port data. However, you need to use it to retrieve master data related to both the port a shipment is sent from and the port it is shipped to. Therefore, you will use a Derived Attribute View.

Creating Derived Attribute Views

Note the following items in relation to Derived Attribute Views:

- They open in read-only mode.
- The only field you can modify is its description.
- They are always strictly identical to the attribute view they are derived from.

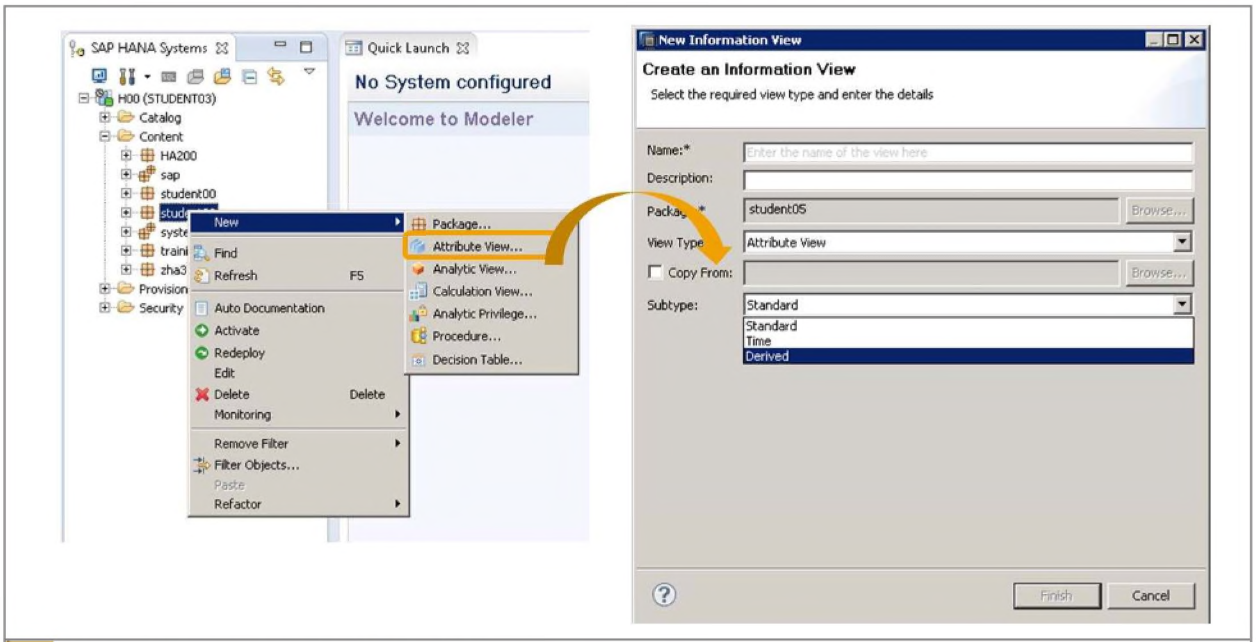


Figure 79: Creating Derived Attribute Views

Scenario in Attribute and Analytic Views

You have learned that all types of graphical Calculation View enable complex calculation scenarios by creating as many nodes as required by your business case.

On the contrary, Attribute and Analytic Views only provide standard and simple calculation scenarios.

Attribute View

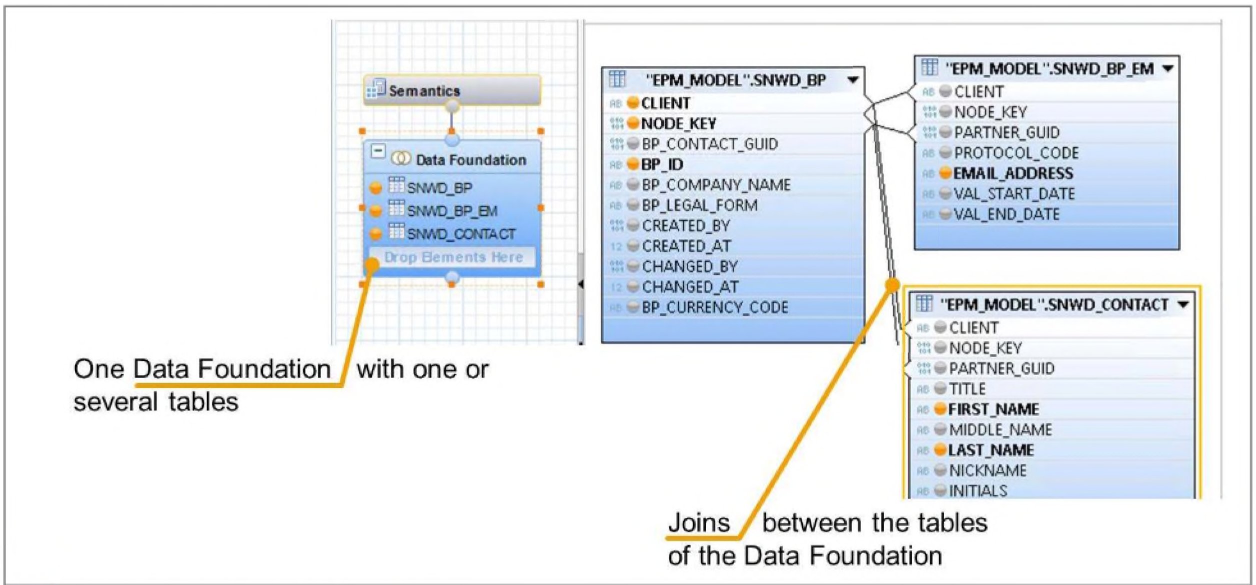


Figure 80: Attribute View Scenario

In Attribute views, one Data Foundation can contain one or several tables. The joins between these tables are defined within the Data Foundation. As it is unique, the Data Foundation node can join more than two tables, if needed.

> Note:
> To create a join between more than two tables in a Calculation View, you can stack join nodes.

If needed, calculated attributes are also defined in the Data Foundation.

Analytic Views

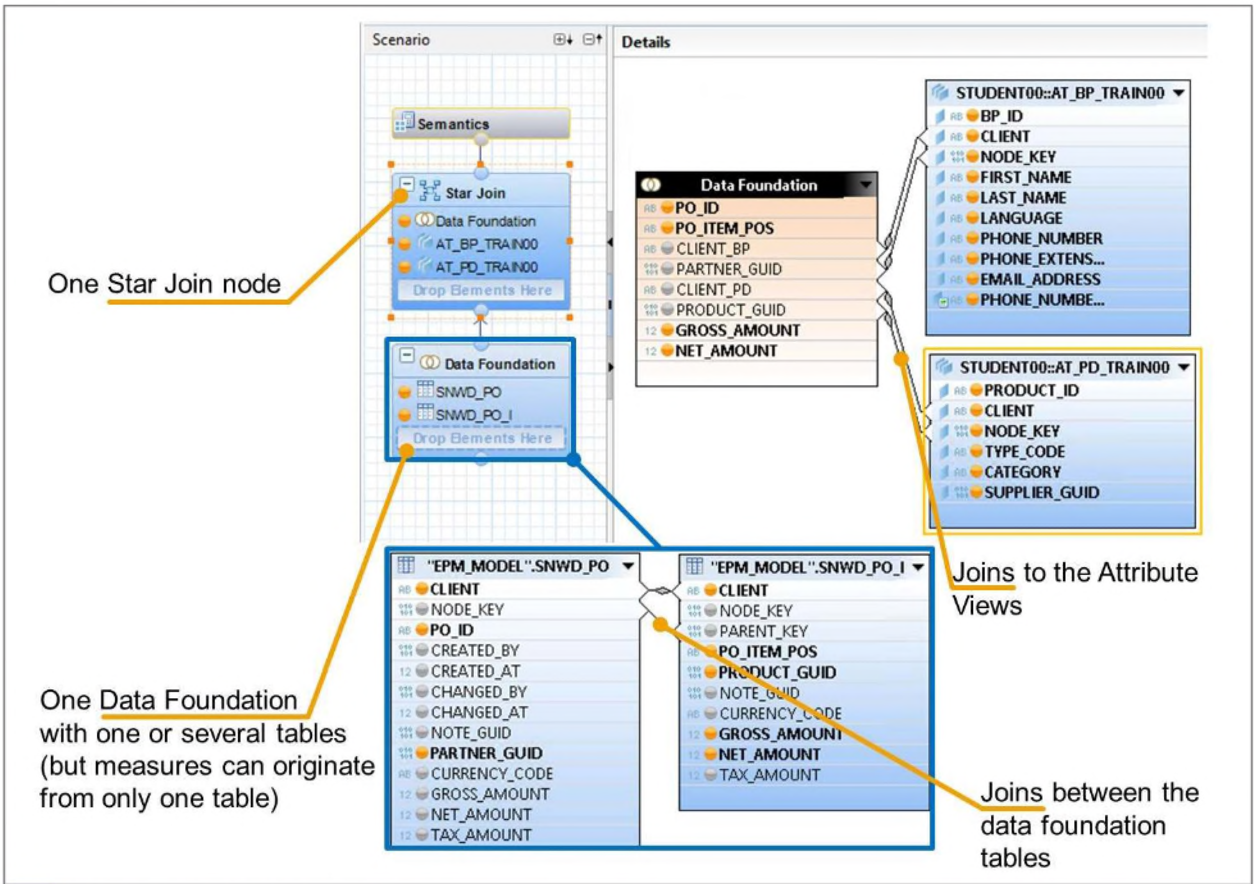


Figure 81: Analytic View Scenario

In Analytic Views, one Data Foundation can include one or several tables, but measures can only originate from one table. This means that some attributes can be provided by other tables of the data foundation, rather than the unique one that provides measures to the attributes provided by the Attribute Views included in the Star Join.

The Star Join of Analytic Views is designed in a similar way to the Calculation Views, except that it supports Attribute Views exclusively.

> Note:
> As a reminder, in a Calculation View of type Cube With Star Join, the Star Join node is also unique, and only supports Calculation Views of type Dimension, in addition to the node that provides fact data.

Data Foundation of Analytic Views

Compared with the *Cube with Star Join* Calculation Views, the Analytic View has two main limitations with regards to its Data Foundation

Functional Limitations in the Data Foundation in Analytic Views

- Limited Types of Data Source

  The Data Foundation of an Analytic View cannot be defined on row tables, or other information models (column views).The main data source type is Column Tables.
- Facts can only originate from one table.

  You cannot select measures from several tables of the Data Foundation.

Note:
In practice, you can build an Analytic View with a Data Foundation made up of several tables (with joins between these tables), but the validation will fail whenever you add to the output measures from more than one table of the Data Foundation.

Calculated Columns in Analytic Views

Calculated columns in Analytic Views are defined at the Star Join level, and apply by default AFTER data has been aggregated according to the columns queried by the client tool.

An option is available to force the calculation before the aggregation.

Join Behavior by Type of Information Views

Join Type	Calculation View behavior	Attribute View behavior	Analytic View behavior
REFERENTIAL	"Optional" join	Inner join	"Optional" join
TEXT	Left outer join	Left outer join	Left outer join
TEMPORAL	Inner join	Not available	Inner or Referential join
SPATIAL	Performs elaborate spatial computation (within, contains…)	Not available	Not available
FULL OUTER	Full outer join (with special restrictions in *Star Join*)	Not available	Not available

Figure 82: Join Behavior in Different Information Views

This figure, Join Behavior in Different Information Views, shows how each type of join behaves, depending on the type of information view where it is used.

Note:
The *Inner*, *Left Outer*, and *Right Outer* are not mentioned here, because they have exactly the same behavior in all the types of Information Views.

LESSON SUMMARY

You should now be able to:

- Explain Attribute Views and Analytic Views (Deprecated View Types)

Learning Assessment

1. When can you use the Referential join?

   Choose the correct answer.

   - [ ] A When you have to report on facts with matching dimensions.
   - [ ] B When you have to report on all posted facts.
   - [ ] C When Referential Integrity is assured.
   - [ ] D When you have to report on all dimensions to confirm if there are matching facts or not.

2. Text Join, which acts as a Right Outer Join, is used to join a text table to a master data table.

   Determine whether this statement is true or false.

   - [ ] True
   - [ ] False

3. Identify the join that checks the From and To Column on the left table.

   Choose the correct answer.

   - [ ] A Left Outer Join
   - [ ] B Right Outer Join
   - [ ] C Temporal Join
   - [ ] D Full Outer Join
   - [ ] E Text Join

4. Which of these information models support unions?

   Choose the correct answers.

   - [ ] A Analytic Views
   - [ ] B Script-Based Calculation Views
   - [ ] C Attribute Views
   - [ ] D Graphical Calculation Views

5. The columns labels you define in a Dimension Calculation View are used internally and are not exposed to client tools.

   Determine whether this statement is true or false.

   - [ ] True
   - [ ] False

6. When do you use the hidden attribute in a Dimension Calculation View?

   Choose the correct answer.

   - [ ] A When you want to hide an attribute that is not required for client consumption.
   - [ ] B When you want to hide the attribute in your CUBE Calculation View.
   - [ ] C When you need to include the attribute in your Calculation View to define a join.

7. Identify the features and benefits of a calculation view.

   Choose the correct answers.

   - [ ] A It can be defined as a graphical view or a scripted view.
   - [ ] B It is not possible to join several fact tables in a calculation view.
   - [ ] C It supports you to include more advanced calculations.
   - [ ] D It can contain Calculated Attributes.

8. What is the function of the Aggregation Node in a Graphical Calculation View?

   Choose the correct answer.

   - [ ] A Map the sources to the target.
   - [ ] B Control the aggregation.
   - [ ] C Combine the result sets.

9. Which of the following features is specific to deprecated graphical Information View types and is not supported by Calculation Views in SAP HANA SPS12?

Choose the correct answer.

- [ ] A Star Join
- [ ] B Fuzzy Search
- [ ] C Union

Learning Assessment - Answers

1. When can you use the Referential join?

   Choose the correct answer.

   - [ ] A When you have to report on facts with matching dimensions.
   - [ ] B When you have to report on all posted facts.
   - [X] C When Referential Integrity is assured.
   - [ ] D When you have to report on all dimensions to confirm if there are matching facts or not.

2. Text Join, which acts as a Right Outer Join, is used to join a text table to a master data table.

   Determine whether this statement is true or false.

   - [ ] True
   - [X] False

3. Identify the join that checks the From and To Column on the left table.

   Choose the correct answer.

   - [ ] A Left Outer Join
   - [ ] B Right Outer Join
   - [X] C Temporal Join
   - [ ] D Full Outer Join
   - [ ] E Text Join

4. Which of these information models support unions?

   Choose the correct answers.

   - [ ] A Analytic Views
   - [x] B Script-Based Calculation Views
   - [ ] C Attribute Views
   - [x] D Graphical Calculation Views

5. The columns labels you define in a Dimension Calculation View are used internally and are not exposed to client tools.

   Determine whether this statement is true or false.

   - [ ] True
   - [x] False

6. When do you use the hidden attribute in a Dimension Calculation View?

   Choose the correct answer.

   - [x] A When you want to hide an attribute that is not required for client consumption.
   - [ ] B When you want to hide the attribute in your CUBE Calculation View.
   - [ ] C When you need to include the attribute in your Calculation View to define a join.

7. Identify the features and benefits of a calculation view.

   Choose the correct answers.

   - [x] A It can be defined as a graphical view or a scripted view.
   - [ ] B It is not possible to join several fact tables in a calculation view.
   - [x] C It supports you to include more advanced calculations.
   - [x] D It can contain Calculated Attributes.

8. What is the function of the Aggregation Node in a Graphical Calculation View?

   Choose the correct answer.

   [ ] A Map the sources to the target.

   [X] B Control the aggregation.

   [ ] C Combine the result sets.

9. Which of the following features is specific to deprecated graphical Information View types and is not supported by Calculation Views in SAP HANA SPS12?

   Choose the correct answer.

   [ ] A Star Join

   [X] B Fuzzy Search

   [ ] C Union

UNIT 2 Modeling Functions

Lesson 1

Lesson 2

Lesson 3

Lesson 4

Lesson 5

Lesson 6

UNIT OBJECTIVES

- Use hierarchies
- Create parent-child hierarchies
- Work with time-based hierarchies

- Create restricted and calculated columns
- Create Restricted Columns
- Create calculated columns
- Filter data
- Create client dependent views
- Restrict data when modeling using domain fix values
- Use variables and input parameters
- Create variables and use them to filter data
- Create input parameters
- Explain the general principles of currency conversion
- Apply currency conversion in Calculation Views
- Create decision tables
- Consume decision tables in Calculation Views

Unit 2
Lesson 1

Using Hierarchies

LESSON OVERVIEW

This lesson explains how to implement hierarchies in information models using SAP HANA studio.

Business Example

Hierarchies are usually used in business intelligence reporting to display characteristics across aggregated nodes.

For example, you may have business requirement to display customers in a geographical organization with country, state, and city in a hierarchy.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Use hierarchies
- Create parent-child hierarchies
- Work with time-based hierarchies

Hierarchy

The type of hierarchy you want to create will depend on the structure of the source data

- A level hierarchy requires each node in a separate column
- A parent child hierarchy requires separate columns for parents and children.

Parent Child Hierarchy source data:

Parent	Child
DACH	Germany
Germany	Bayern
Germany	Hamburg
EMEA	Ireland
Ireland	Kerry

Level Hierarchy source data:

Region	Country	State
DACH	Germany	Bayern
DACH	Germany	Hamburg
EMEA	Ireland	Kerry

Figure 83: Choosing Hierarchy Type

> Note:
> Parent-child hierarchy columns usually contain IDs or key fields instead of plain text.

Hierarchy Comparison

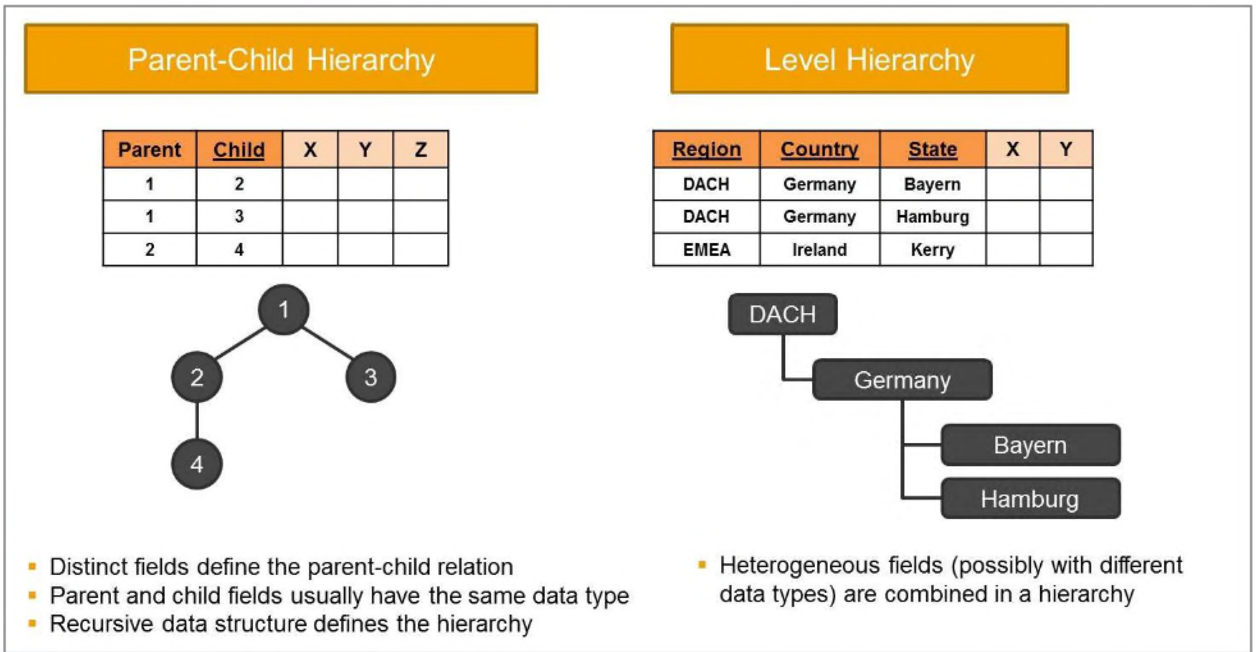


Figure 84: Hierarchy Comparison

Level Hierarchies

A level hierarchy is rigid in nature, and the root and child nodes can only be accessed in a defined order.

To implement level hierarchies, use the following procedure:

- Select the source table(s) for the view :

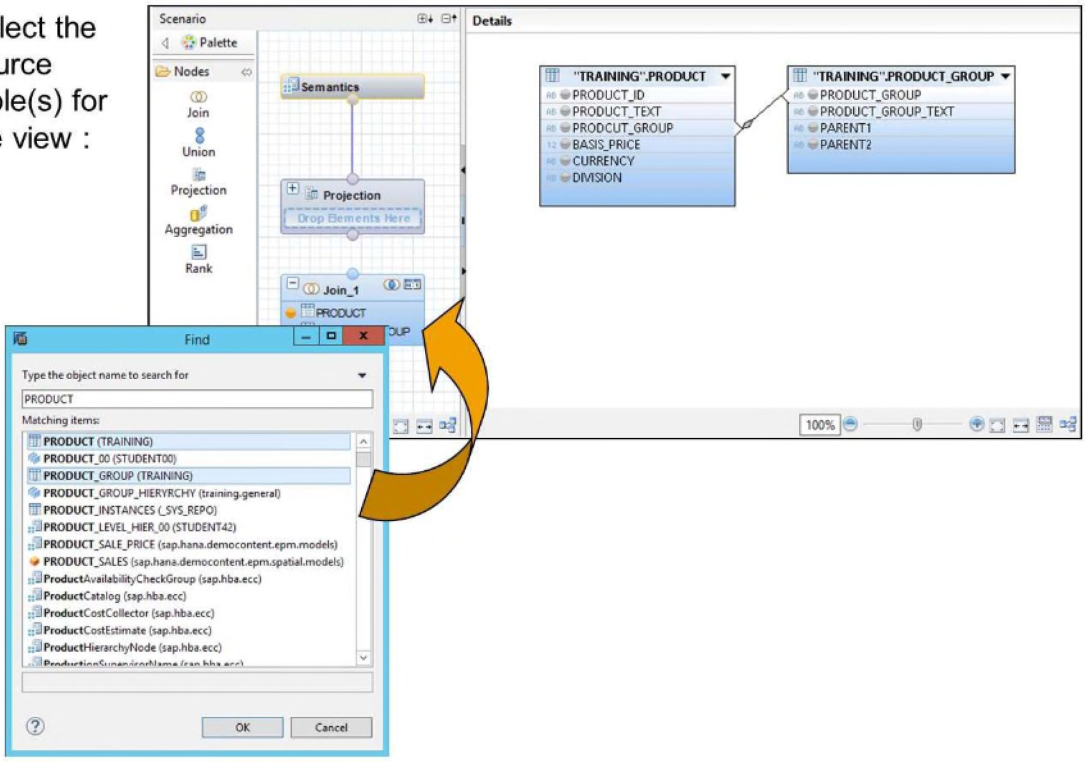


Figure 85: Implement Level Hierarchies (I)

Level Hierarchies (2)

- Select the columns that should be part of the view, including any columns required for the hierarchy:

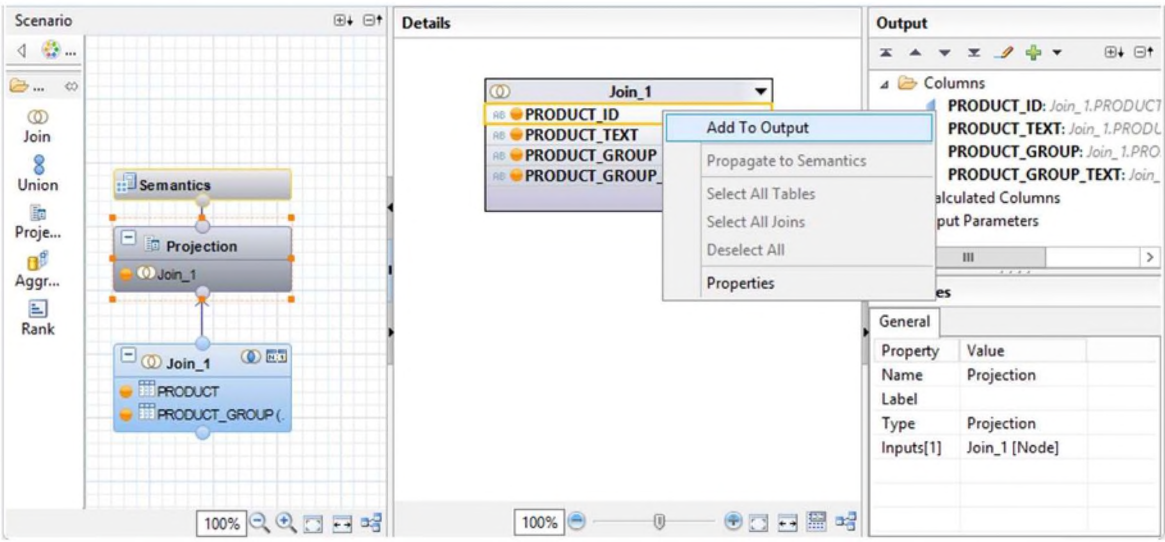


Figure 86: Implement Level Hierarchies (II)

Level Hierarchies (3)

- In the Semantics node, select the Hierarchies tab and click the '+' button in the Hierarchy pane:

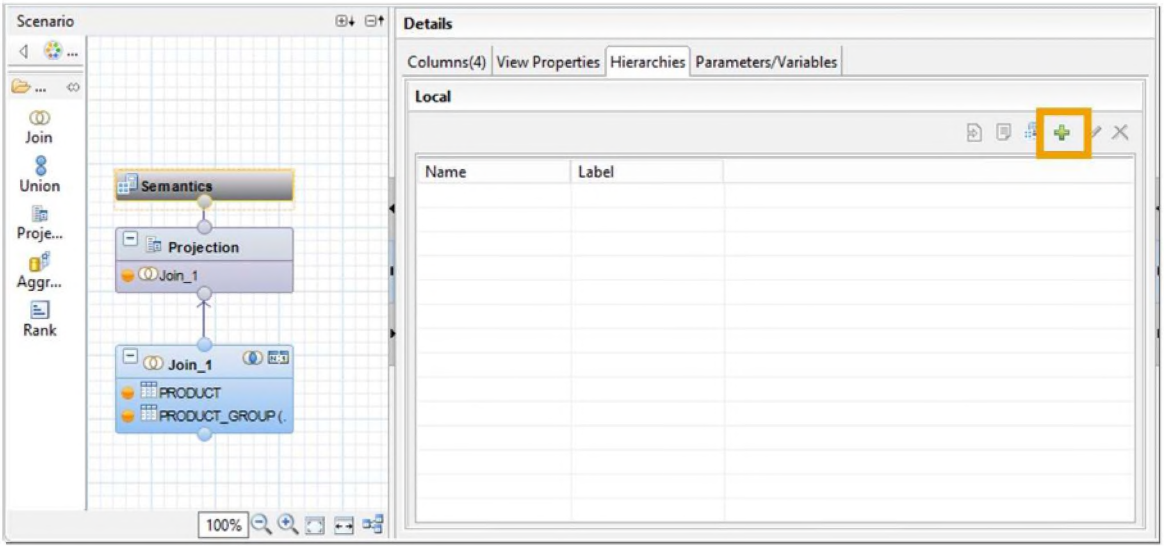


Figure 87: Implement Level Hierarchies (III)

Level Hierarchies (4)

- Add the columns to the hierarchy in the correct level order from top to bottom, with the lowest granularity at the lowest level of the hierarchy.
- Additionally, you can define an ascending or descending sort direction per level.

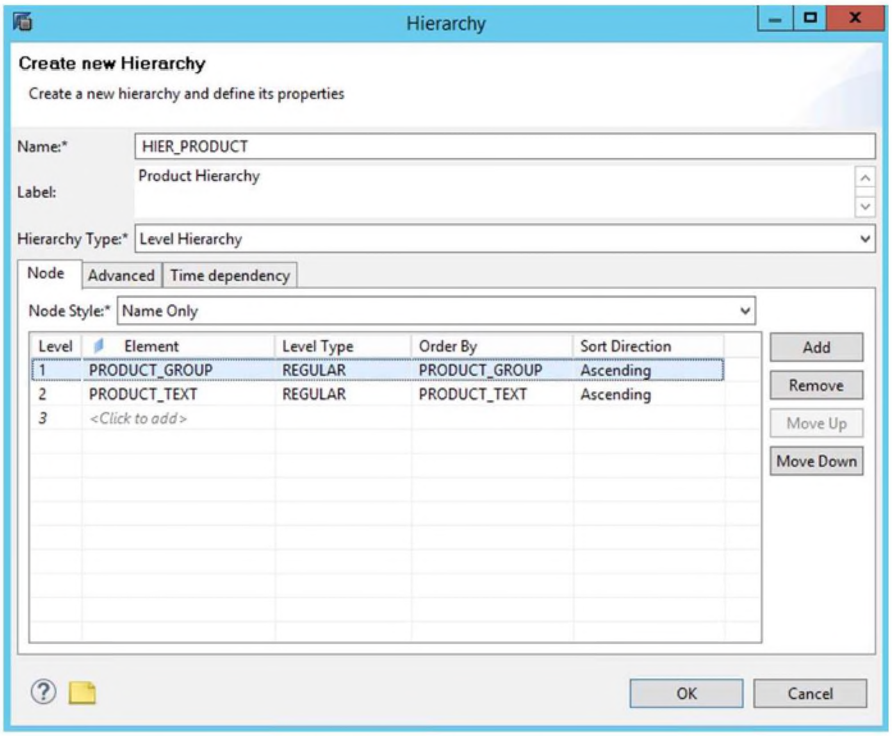


Figure 88: Implement Level Hierarchies (IV)

Note:
When you preview a calculation view containing hierarchies using SAP HANA studio, you will not be able to see the hierarchy in the same way it is displayed when you use a reporting tool supporting hierarchies.

Node Styles

Node styles are used to define the output format of a node ID.

Using a fiscal hierarchy example, the following table demonstrates the different node styles:

Table 6: Node Styles

Level Style	Output	Example
Level Name	Level and node name	MONTH.JAN
Name Only	Node name only	JAN
Name Path	Node name and its ancestors	FISCAL_2015.QUARTER_1.JAN

Level Types

Level Type specifies the semantics for the level attributes. For example, the level type, *TIMEMONTHS*, indicates that the attributes are months such as January, February, or March.

The level type, *REGULAR*, indicates that the level does not require any special formatting.

Hierarchy Member Order

Using the *Order By* drop-down list, an attribute can be selected for ordering the hierarchy members in the order specified in the *Sort Direction* column.

Parent Child Hierarchies

When creating a parent-child hierarchy, the first step is to define the nodes making up the hierarchy.

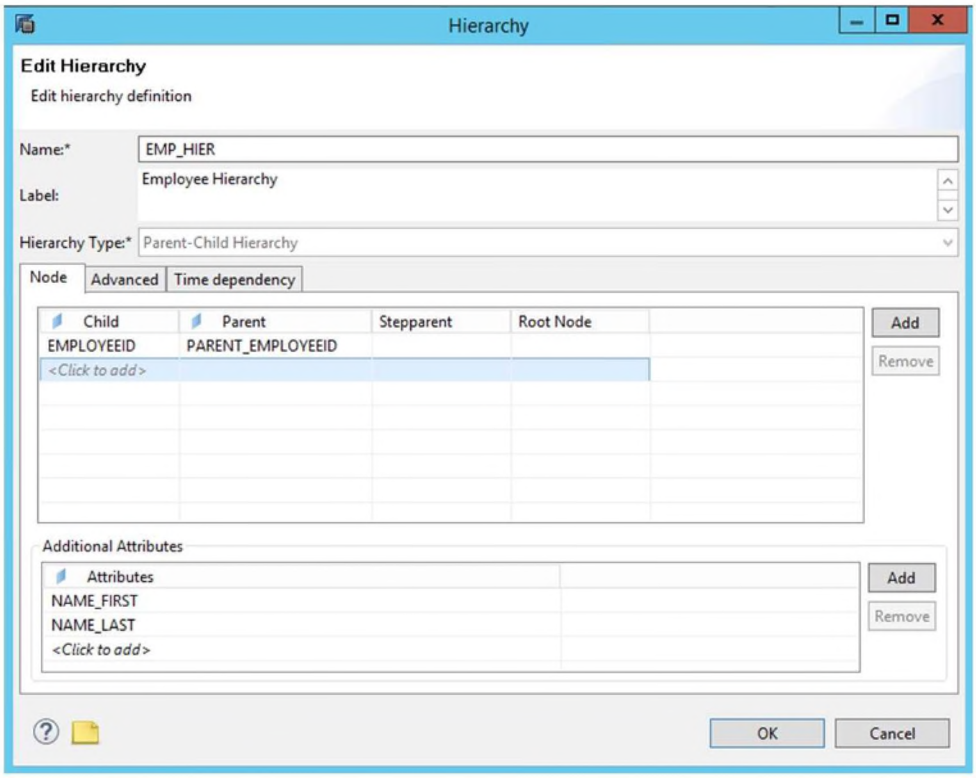


Figure 89: Defining the Nodes of a Parent-Child Hierarchy

The *Child* column contains the attribute used as the child within the hierarchy, whereas the *Parent* column contains the attribute used for its parent.

You can define multiple parent child pairs to support the compound node IDs. For example:

- *CostCenter → ParentCostCenter*
- *ControllingArea → ParentControllingArea*

The preceding list of parent child pairs constitutes a compound parent-child definition to uniquely identify cost centers.

Caution:
Multiple parents and compound parent-child definitions are not supported by MDX.

Advanced Properties of a Parent-Child Hierarchy

Additional attributes can also be added to the hierarchy making it easier to report on.

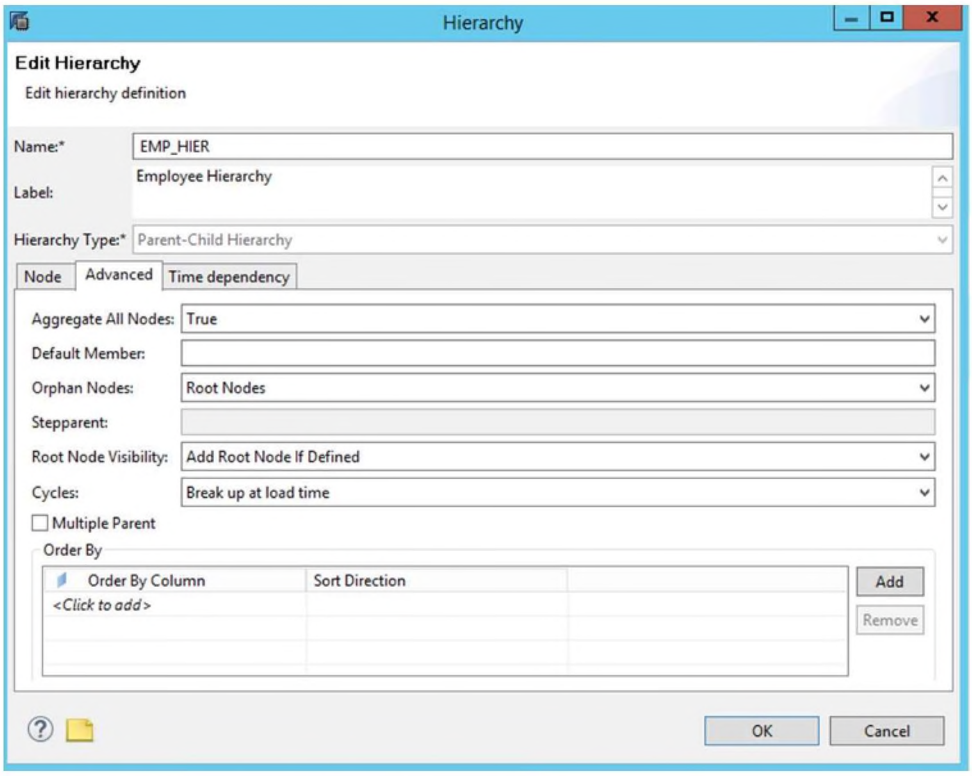


Figure 90: Advanced Properties of a Parent-Child Hierarchy

Aggregate All Nodes

The *Aggregate All Nodes* property defines whether the values of intermediate nodes of the hierarchy should be aggregated to the total value of the hierarchy's root node. If you are sure that there is no data posted on aggregate nodes, you should set the option to *False*. The engine then executes the hierarchy faster.

> Note:
> The value of the *Aggregate All Nodes* property is interpreted only by the SAP HANA MDX engine.

Default Member

The *Default Member* value helps identify the default member of the hierarchy. If you do not provide any value, all members of the hierarchy are default members.

Orphan Nodes

In a parent-child hierarchy, you might encounter orphan nodes without a parent. The *Orphan Nodes* property defines how these should be handled as follows:

- Root Nodes

  Any orphan node will be defined as a hierarchy root node.

- Error

  When encountering an orphan node, the view will throw an error.

- Ignore

  Orphan nodes will be ignored.

- Stepparent

  Using the Stepparent option, orphan nodes will fall under the stepparent node ID as defined on the *Node* tab.

> Caution:
> If you choose to assign an orphan node to a stepparent, the following rules apply:
>
> - The stepparent node must be already defined in the hierarchy at the ROOT level.
> - The stepparent ID must be entered according to the node style defined in the hierarchy.

Root Node Visibility

The *Root Node Visibility* property is used to define if an additional root node needs to be added to the hierarchy.

Cycles

Cycles are typically not desired in a hierarchy.

In such cases, the *Cycles* property is used to define how these should be broken when encountered, or whether an error should be thrown.

Time Dependent Hierarchies

Time dependency is supported for more complex hierarchy data, such as human resources applications with their organizations, or material management systems with BOMs where information is reliant on time.

> Caution:
> Defining a time dependency is only possible in Calculation Views, and for hierarchies of type Parent-Child.

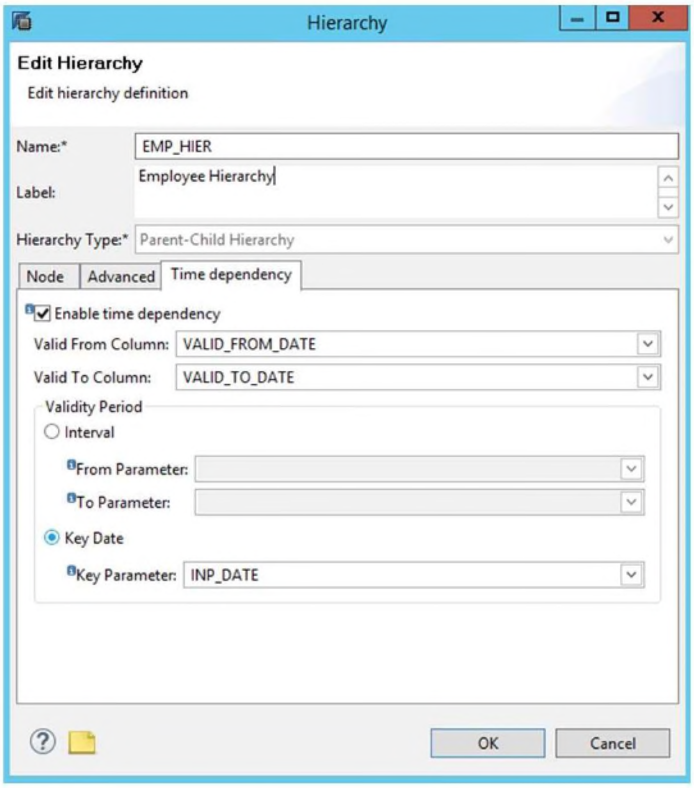


Figure 91: Defining Time Dependent Hierarchies

Enabling time dependency supports hierarchies based on changing elements valid for specific time periods. This allows displaying different versions of a hierarchy.

Your source data needs to contain definition columns consisting of a *Valid From* and a *Valid To* column.

Determining the Validity Period

When a hierarchy needs to show elements from an interval, you need to define two input parameters, a *From Parameter*, and a *To Parameter*. If the hierarchy needs to show elements valid on a specific date, you need one input parameter defined as the *Key Date*.

Drill Down Enablement

By default, MDX only shows key fields, which are governed by an output field property of the attribute view in the Data Foundation.

If the *Drill Down Enablement* property is left as default for a non-key field, this field will not show up for reporting in an MDX client such as Microsoft Excel.

- By setting Drill Down Enablement to 'Drill Down with flat hierarchy (MDX)', you can make sure the column can be used for reporting using MDX, even though it is not a key field.

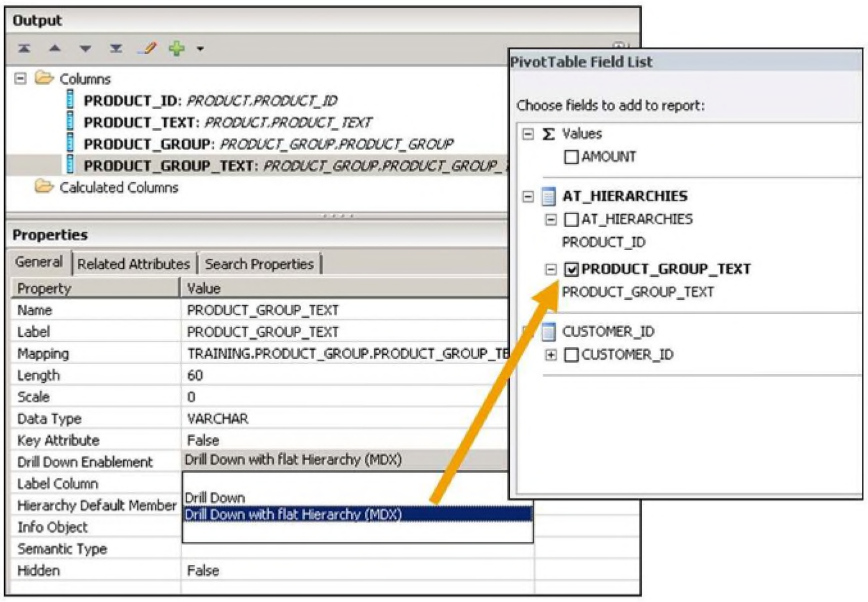


Figure 92: Drill Down Enablement in MDX

When *Drill Down Enablement* is set to *Drill Down with flat hierarchy (MDX)* the attribute is enabled for drill-down and an additional flat hierarchy is generated.

In the generated flat hierarchy, all the distinct attribute values make up the first and only level of the hierarchy.

Enabling Hierarchies for SQL Access

A hierarchy defined in a Dimension Calculation View is available (as a shared hierarchy) in any Calculation View of type Cube with Star Join that reference the Dimension view in its Star Join. SAP HANA enables you to include a specific column for this hierarchy, which you can query by SQL. This column will display the different nodes of the hierarchy enabling filtering, or group by, in SQL queries executed against the CUBE with Star Join view.

> Note:
> This column is only accessible via SQL. It is not exposed to the graphical information models that consume the view.

For a given Cube with Star Join Calculation View, the following two options enable SQL Access to the shared hierarchies defined in the underlying Dimension views:

CUBE with Star Join Calculation View – Enabling SQL Access to Hierarchies

- Option 1: Enable all shared hierarchies at once.

  You can enable SQL access to all of the shared hierarchies defined in the various Dimension Calculation Views.

  In the *View Properties* pane, select the *Enable Hierarchies for SQL access* checkbox.

- Option 2: Select which shared hierarchies you want to enable.

  You can specify which of the shared hierarchies you want to enable for SQL access.

In the *Hierarchies* pane, double-click a shared hierarchy and in the *SQL Access* tab, select the *Enable SQL access* checkbox.

You will learn how to consume the Hierarchy Columns with SQL later on, in the SQL Script and Procedures unit.

Overview of the SAP BI Tools that Support SAP HANA Hierarchies

There are a number of client tools in the SAP and SAP BusinessObjects Portfolio that support hierarchies.

The figure, SAP BI Tools Support for Hierarchies, shows the current status for SAP HANA SPS12 and SAP BusinessObjects BI 4.2.

Client Tool	Parent-Child Hierarchies	Level Hierarchies
Analysis, Edition for Office (Excel)	✓	✓
Analysis, Edition for OLAP	✓	✓
Crystal Report for Enterprise	✓	✓
Lumira, Predictive Analysis		✓
Design Studio	✓	✓

Figure 93: SAP BI Tools Support for Hierarchies

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

Unit 2
Exercise 7

Implement a Level Hierarchy

Business Example

You have been asked to build a hierarchy for products that will be used to analyze Sales Order data.

> Note:
> In this exercise, when values include ##, replace the characters with the number your instructor provided to you.

1. Create a new calculation view using the data in the following table:

Field	Value
Name	**CVC_LEV_HIER_##**
Label	**Products Level Hierarchy**
Package	*STUDENT##*
View Type	*Calculation View*
Subtype	*Standard*
Type	*Graphical*
Data Category	*CUBE*

2. In the calculation view scenario, under the *Aggregation* node, add a Join node.

> Note:
> Leave some space available between the *Aggregation* node and the *Join* node that you create, as you will later add an additional node in-between.

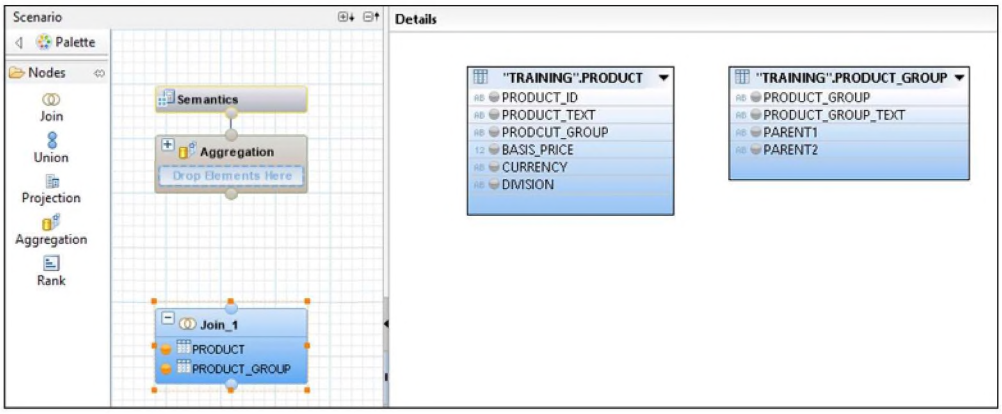


Figure 94: Add a Join Node

3. Add the following tables to the *Join* node:

- **TRAINING.PRODUCT**
- **TRAINING.PRODUCT_GROUP**

4. In your *Join* node, create a join between the tables:

Field	Value
Left Table	TRAINING.PRODUCT
Right Table	TRAINING.PRODUCT_GROUP
Join Columns	PRODUCT_GROUP = PRODUCT_GROUP
Join Type	*Left Outer*
Cardinality	*n..1*

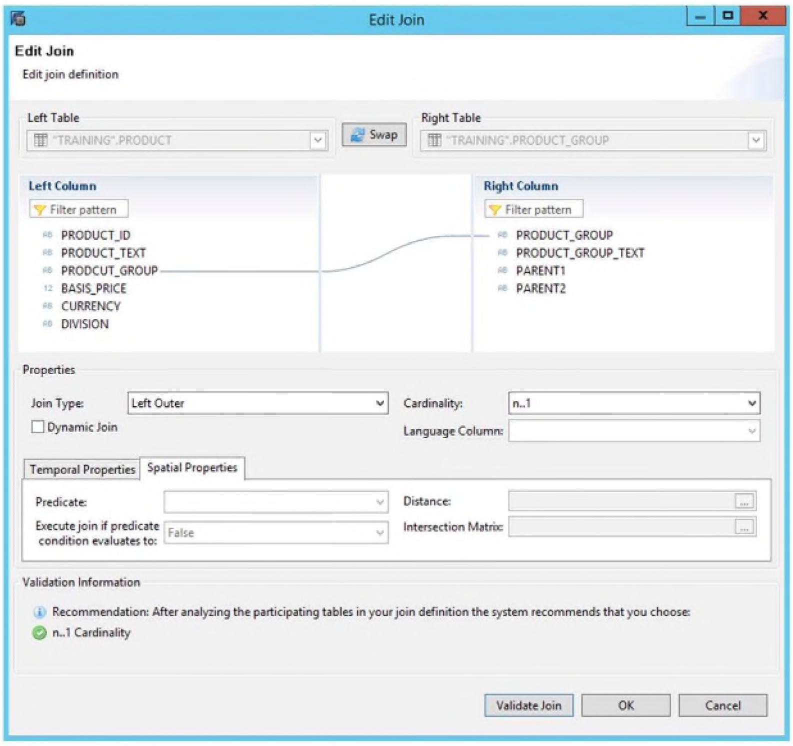


Figure 95: Join Properties

5. In the *Join* node, add the following columns to output:
    - PRODUCT.PRODUCT_ID
    - PRODUCT.PRODUCT_TEXT
    - PRODUCT_GROUP.PRODUCT_GROUP_TEXT
6. Rename the *Join* node as: **PRODUCTS**
7. In the Calculation View scenario, above the *Join* node, add an additional Join node.
8. Rename the new *Join* node as: **SALES**.
9. Add the following table to the new *Join* node:
    - TRAINING.SALES_DATA
10. Connect the *PRODUCT* Join node to the *SALES* join node.
11. In your *SALES* Join node, create a join between the tables:

Field	Value
Left Table	TRAINING.SALES_DATA
Right Table	PRODUCT (node)
Join Columns	PRODUCT_ID = PRODUCT_ID
Join Type	Left Outer
Cardinality	n..1

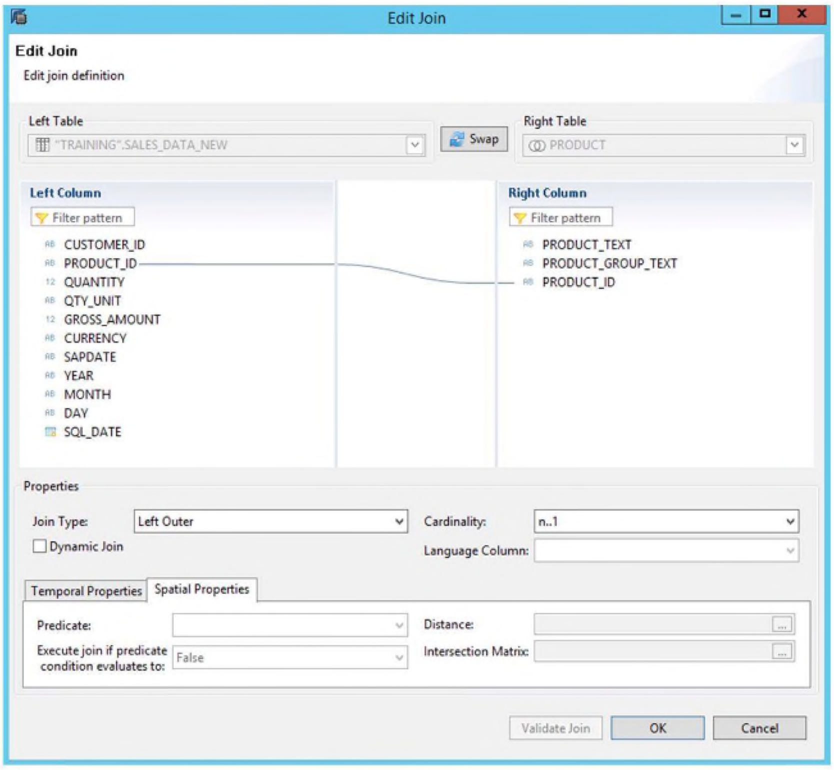


Figure 96: Join Properties

12. In the *Join* node, add the following columns to output:

- PRODUCT.PRODUCT_ID
- PRODUCT.PRODUCT_TEXT
- PRODUCT.PRODUCT_GROUP_TEXT
- SALES_DATA.QUANTITY

> Note:
> Instead of adding the columns from the PRODUCT table individually, you can right-click the *PRODUCT* table and select *Add All To Output* from the context menu.

13. Connect the *SALES* Join node to the *Aggregation* node.
14. In the *Aggregation* node, add the following columns to output:
    - PRODUCT_ID
    - PRODUCT_TEXT
    - PRODUCT_GROUP_TEXT
15. In the *Aggregation* node, add the following column as an aggregated column:
    - QUANTITY
16. In the *Semantics* node, Auto-Assign columns as Attributes and Measures.

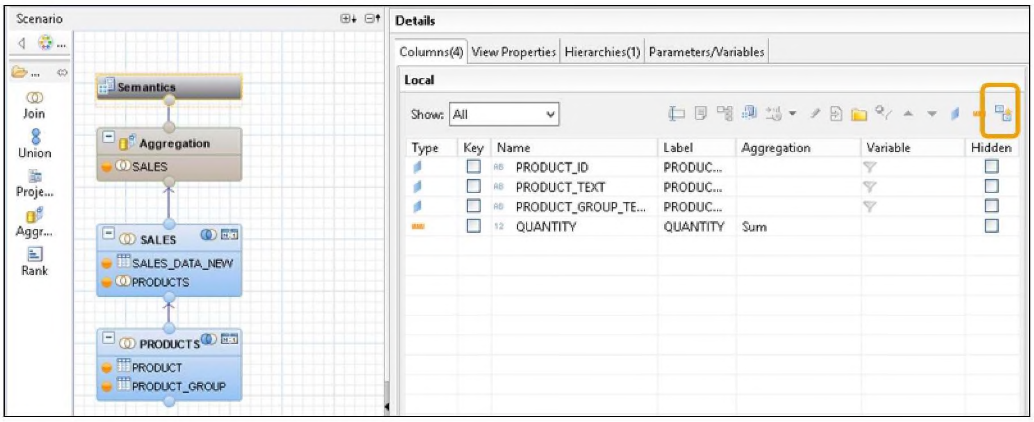


Figure 97: Auto-Assign Measures and Attributes

17. In the *Hierarchy* tab of the *Semantics* node, create a new hierarchy with the following properties:

Field	Value
Name	PRODUCT_LEVEL_HIERARCHY
Label	Product Level Hierarchy
Hierarchy Type	Level Hierarchy
Level 1 Element	PRODUCT_GROUP_TEXT

Field	Value
Level 2 Element	PRODUCT_TEXT

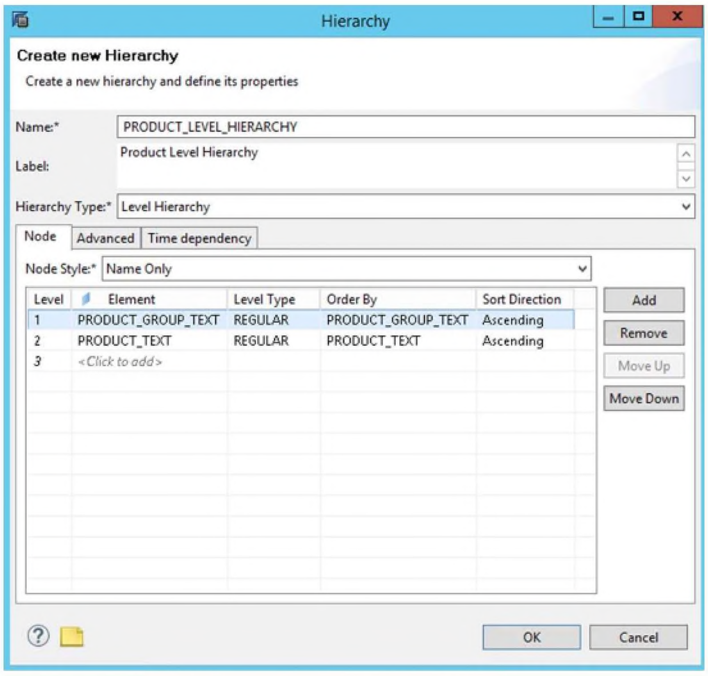


Figure 98: Level Hierarchy Properties

18. Save and Activate the calculation view.

> Note:
> If you preview the data of the calculation view, you will not see the hierarchy that you have created, as SAP HANA Studio does not support hierarchies in preview mode.

Task: Consume the Hierarchy with Microsoft Excel

Use the SAP HANA MDX Client with Microsoft Excel to query your Calculation View data and explore the simple *PRODUCT* hierarchy that you have implemented.

1. Start Microsoft Excel and create a new workbook based on a blank template.
2. Add a new data source based on the *CVC_LEV_HIER_##* Calculation View as a PivotTable Report in cell A1. You need to use the Data Connection Wizard and connect Excel to your calculation view with the following credentials:

Field	Value
Host	**wdflbmt7215.wdf.sap.corp**
Instance Number	**00**
Database Mode	*Single Database*
User	**STUDENT##**
Password	**Training1**

3. Add the following fields to the pivot table:
   - QUANTITY
   - Product Level Hierarchy

> Note:
> In this simple example, the data model does not include the individual product references, and the *PRODUCT_TEXT* column is actually a grouping level in the hierarchy (such as *Keyboards* or *Mouses*, and it corresponds to the *PRODUCT_ID* column.
>
> If you add the *PRODUCT_ID column* to the pivot table, you will not get an additional level of detail. You could have defined the *PRODUCT_TEXT* column as the label column for *PRODUCT_ID*.

4. Filter the pivot table to display only the product groups *Devices* and *Disks*.
5. Close the workbook containing the pivot table without saving it and exit Microsoft Excel.

Unit 2
Solution 7

Implement a Level Hierarchy

Business Example

You have been asked to build a hierarchy for products that will be used to analyze Sales Order data.

> Note:
> In this exercise, when values include ##, replace the characters with the number your instructor provided to you.

1. Create a new calculation view using the data in the following table:

Field	Value
Name	**CVC_LEV_HIER_##**
Label	**Products Level Hierarchy**
Package	*STUDENT##*
View Type	*Calculation View*
Subtype	*Standard*
Type	*Graphical*
Data Category	*CUBE*

   a) From the context menu of your package, in the *Content* node, select *New → Calculation View*.

   b) Enter the view properties as provided in the table.

   c) Choose *Finish*.

2. In the calculation view scenario, under the *Aggregation* node, add a Join node.

> Note:
> Leave some space available between the *Aggregation* node and the *Join* node that you create, as you will later add an additional node in-between.

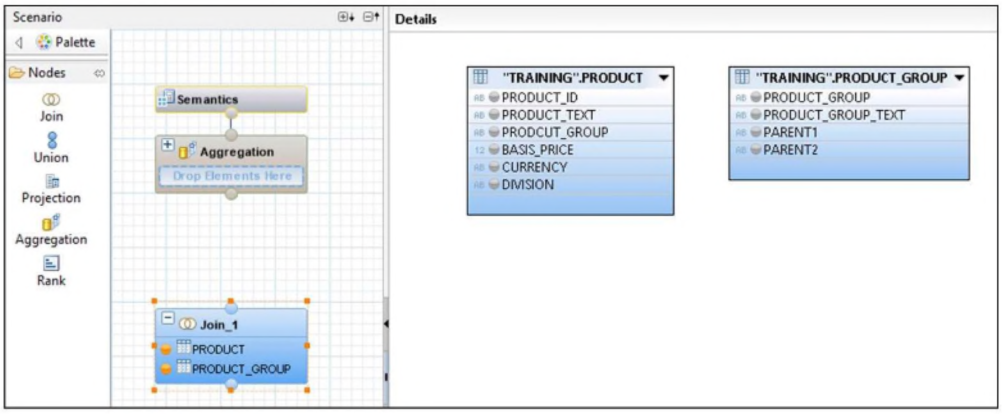


Figure 94: Add a Join Node

a) In the *Scenario* Palette, choose the *Join* node.

b) In the *Scenario*, place a new *Join* node under the *Aggregation* node.

3. Add the following tables to the *Join* node:

- **TRAINING.PRODUCT**
- **TRAINING.PRODUCT_GROUP**

a) Drag the above tables to the *Join* node from the TRAINING schema of the *Catalog* node.

4. In your *Join* node, create a join between the tables:

Field	Value
Left Table	TRAINING.PRODUCT
Right Table	TRAINING.PRODUCT_GROUP
Join Columns	PRODUCT_GROUP = PRODUCT_GROUP
Join Type	*Left Outer*
Cardinality	*n..1*

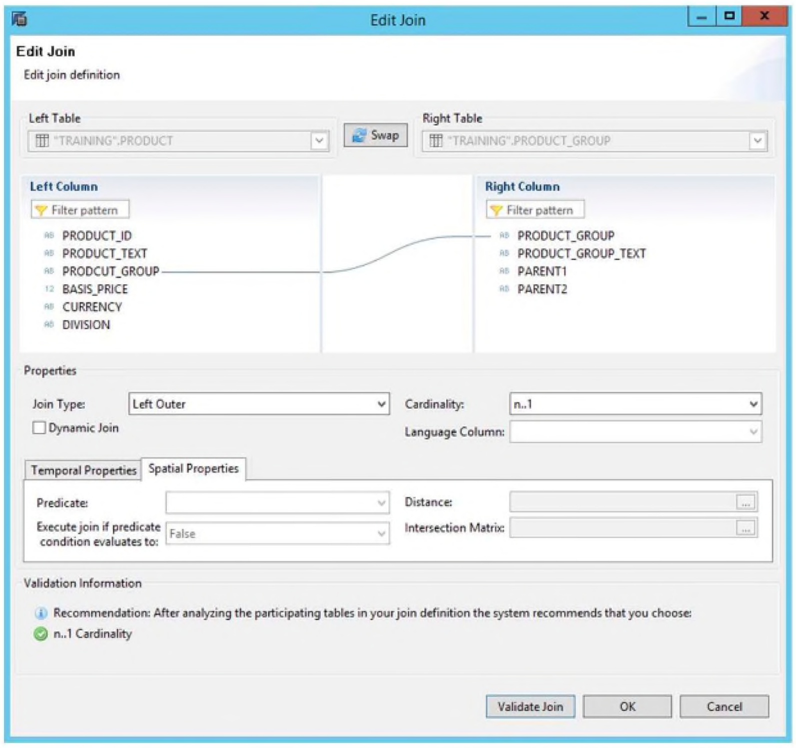


Figure 95: Join Properties

a) Drag a connecting line between the columns *TRAINING.PRODUCT.PRODUCT_GROUP* and *TRAINING.PRODUCT_GROUP.PRODUCT_GROUP*.

b) Double-click the join connector and define the properties as shown in the table.

5. In the *Join* node, add the following columns to output:

   - PRODUCT.PRODUCT_ID
   - PRODUCT.PRODUCT_TEXT
   - PRODUCT_GROUP.PRODUCT_GROUP_TEXT

   a) In the *Join* node, right-click the above columns and select *Add To Output* from the context menu.

6. Rename the *Join* node as: **PRODUCTS**

   a) Right-click the *Join* node and select *Rename* from the context menu.

   b) Enter **PRODUCTS**, and choose *OK*.

7. In the Calculation View scenario, above the *Join* node, add an additional Join node.

   a) In the *Scenario* Palette, choose the *Join* node.

   b) In the *Scenario*, place a new Join node above the existing *Join* node.

8. Rename the new *Join* node as: **SALES**.

   a) Right-click the new Join node and select *Rename* from the context menu.

   b) Enter **SALES**, and choose *OK*.

9. Add the following table to the new *Join* node:

   - TRAINING.SALES_DATA

a) Drag the above table to the *Join* node from the TRAINING schema of the *Catalog* node.

10. Connect the *PRODUCT* Join node to the *SALES* join node.

 a) In the Scenario of the calculation view, drag a connecting line from the *PRODUCT* Join node to the *SALES* join node.

11. In your *SALES* Join node, create a join between the tables:

Field	Value
Left Table	TRAINING.SALES_DATA
Right Table	PRODUCT (node)
Join Columns	PRODUCT_ID = PRODUCT_ID
Join Type	Left Outer
Cardinality	n..1

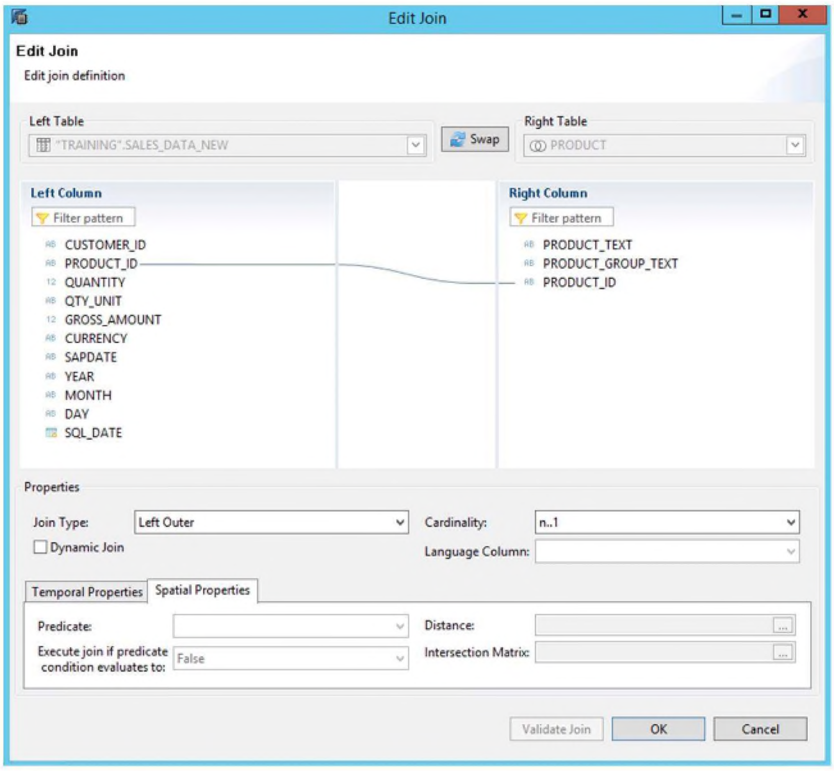


Figure 96: Join Properties

 a) Drag a connecting line between the columns TRAINING.SALES_DATA.PRODUCT_ID and PRODUCT.PRODUCT_ID.

 b) Double-click the join connector and define the properties as per the above table.

12. In the *Join* node, add the following columns to output:

- PRODUCT.PRODUCT_ID
- PRODUCT.PRODUCT_TEXT
- PRODUCT.PRODUCT_GROUP_TEXT
- SALES_DATA.QUANTITY

a) In the *Join* node, right-click the above columns and select *Add To Output* from the context menu.

> Note:
> Instead of adding the columns from the PRODUCT table individually, you can right-click the *PRODUCT* table and select *Add All To Output* from the context menu.

13. Connect the *SALES* Join node to the *Aggregation* node.

    a) In the Scenario of the calculation view, drag a connecting line from the *SALES* Join node to the *Aggregation* node.

14. In the *Aggregation* node, add the following columns to output:

    - PRODUCT_ID
    - PRODUCT_TEXT
    - PRODUCT_GROUP_TEXT

    a) In the *Aggregation* node, right-click the above columns and select *Add To Output* from the context menu.

15. In the *Aggregation* node, add the following column as an aggregated column:

    - QUANTITY

    a) In the *Aggregation* node, right-click the above column and select *Add As Aggregated Column* from the context menu.

16. In the *Semantics* node, Auto-Assign columns as Attributes and Measures.

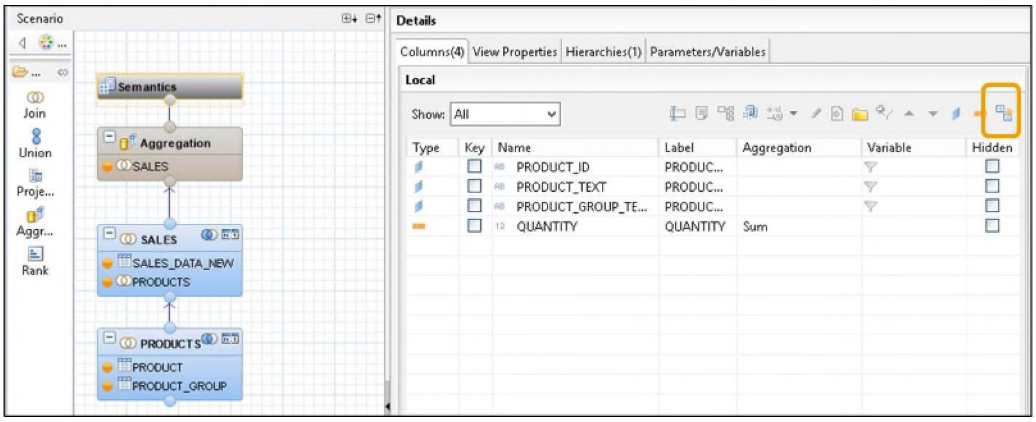


Figure 97: Auto-Assign Measures and Attributes

    a) In the *Columns* tab of the *Semantics* node, choose *Auto-Assign*.

17. In the *Hierarchy* tab of the *Semantics* node, create a new hierarchy with the following properties:

Field	Value
Name	PRODUCT_LEVEL_HIERARCHY
Label	Product Level Hierarchy

Field	Value
Hierarchy Type	Level Hierarchy
Level 1 Element	PRODUCT_GROUP_TEXT
Level 2 Element	PRODUCT_TEXT

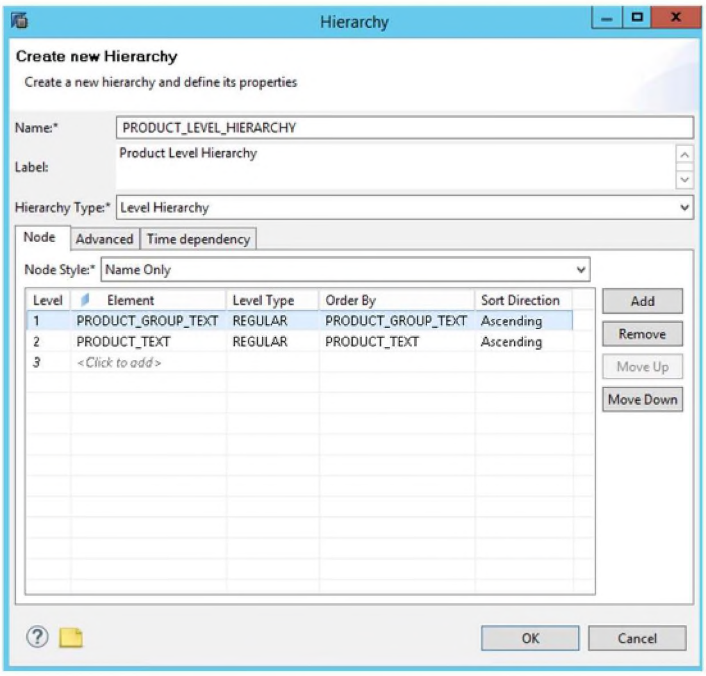


Figure 98: Level Hierarchy Properties

a) In the *Semantics Node*, select the *Hierarchies* tab.

b) Choose + to create a new hierarchy.

c) Enter the *Name* and *Label* as provided in the table.

d) On the *Node* tab, in the *Element* column, choose the text *<Click to Add>* and select attribute *PRODUCT_GROUP_TEXT* as *Level 1* and attribute *PRODUCT_TEXT* as *Level 2*.

e) Choose *OK*.

18. Save and Activate the calculation view.

> Note:
> If you preview the data of the calculation view, you will not see the hierarchy that you have created, as SAP HANA Studio does not support hierarchies in preview mode.

Task: Consume the Hierarchy with Microsoft Excel

Use the SAP HANA MDX Client with Microsoft Excel to query your Calculation View data and explore the simple *PRODUCT* hierarchy that you have implemented.

1. Start Microsoft Excel and create a new workbook based on a blank template.

   a) In the Windows *Start* page, choose *Excel 2013*.

   b) Choose the *Blank workbook* template.

2. Add a new data source based on the *CVC_LEV_HIER_##* Calculation View as a PivotTable Report in cell A1. You need to use the Data Connection Wizard and connect Excel to your calculation view with the following credentials:

Field	Value
Host	**wdflbmt7215.wdf.sap.corp**
Instance Number	**00**
Database Mode	*Single Database*
User	**STUDENT##**
Password	**Training1**

a) Choose *Data* → *From Other Sources* → *From Data Connection Wizard*.

b) Select *Other/Advanced* and choose *Next*.

c) Select *SAP HANA MDX Provider* and choose *Next*.

d) Enter the logon info and credentials as per the table.

e) choose *Test Connection* and choose *OK*.

If you get a connection error, double-check the credentials and test the connection again.

f) At the *Select Database and Table* stage, in the drop-down list, select your *STUDENT##* package and select the *Connect to a specific cube* checkbox.

g) In the list of views, select the view *CVC_LEV_HIER_##* and choose *Finish*.

> Note:
> If you repeat this exercise later and try to add the data source again after modifying it, you might get a dialog box saying that you already have a *CV_LEV_HIER_##.odc* (Office Data Connection) file and asking if you want to replace it. Choose *Yes*.

h) In the *Import Data* dialog box, keep the default options and choose *OK*.

You might need to enter your SAP HANA password again.

3. Add the following fields to the pivot table:

- QUANTITY
- Product Level Hierarchy

> Note:
> In this simple example, the data model does not include the individual product references, and the *PRODUCT_TEXT* column is actually a grouping level in the hierarchy (such as *Keyboards* or *Mouses*, and it corresponds to the *PRODUCT_ID* column.
>
> If you add the *PRODUCT_ID column* to the pivot table, you will not get an additional level of detail. You could have defined the *PRODUCT_TEXT* column as the label column for *PRODUCT_ID*.

a) In the *Pivot Table Fields* panel, select the *QUANTITY* and *Product Level Hierarchy* items.

b) Explore the *Product Level Hierarchy* with the expand/collapse buttons in column A.

4. Filter the pivot table to display only the product groups *Devices* and *Disks*.

a) In, the *Pivot Table Fields* pane, select the arrow on the right of the *PRODUCT_TEXT* column.

b) Select the *(Select All) checkbox.*

c) Select the *Devices* and *Disks* checkboxes.

5. Close the workbook containing the pivot table without saving it and exit Microsoft Excel.

a) Choose *File* → *Close*.

b) Choose *Don't Save*.

c) To exit Microsoft Excel, choose *File* → *Close*.

Unit 2
Exercise 8

Implement a Parent-Child Hierarchy

Business Example

You have been asked to build an Employee hierarchy.

The source data is available as a parent-child hierarchy.

> Note:
> In this exercise, when values include ##, replace the characters with the number your instructor provided to you.

1. Create a new calculation view using the data in the following table:

Field	Value
Name	**`CV_EMPLOYEE_HIER_TRAIN##`**
Label	**`Employee Hierarchy`**
Package	*STUDENT##*
View Type	*Calculation View*
Subtype	*Standard*
Type	*Graphical*
Data Category	*CUBE*

2. In the calculation view scenario, under the *Aggregation* node, add a *Join* node.
3. Add the following tables to the *Join* node:
   - TRAINING.EMPLOYEE
   - TRAINING.EMPLOYEE_HIERARCHY
4. In your *Join* node, create a join between the tables:

Field	Value
Left Table	TRAINING.EMPLOYEE
Right Table	TRAINING.EMPLOYEE_HIERARCHY
Join Columns	EMPLOYEE.DNUMBER = EMPLOYEE_HI-ERARCHY.CHILD_DNUMBER
Join Type	Left Outer

Field	Value
Cardinality	1..n

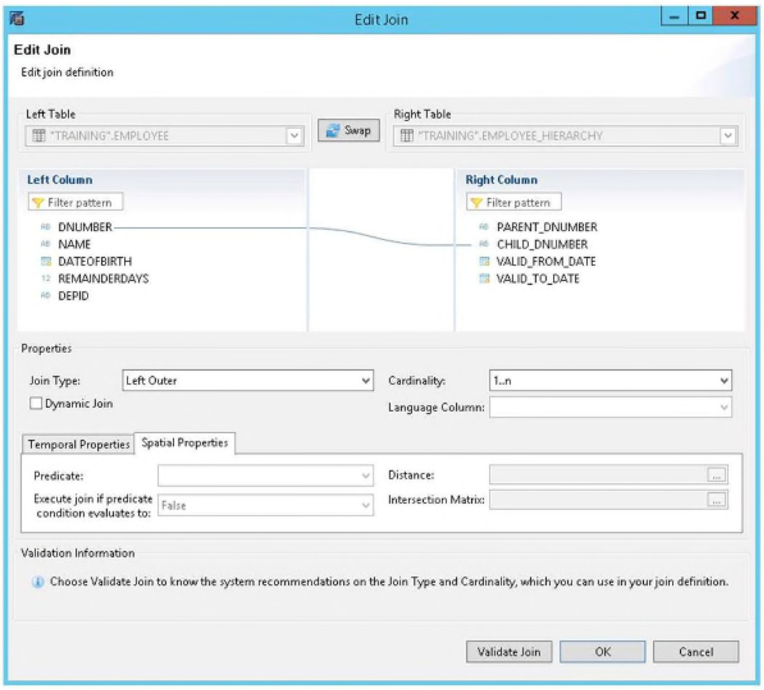


Figure 99: Join of EMPLOYEE and EMPLOYEE_HIERARCHY

5. In the *Join* node, add the following columns to output:

   - EMPLOYEE.DNUMBER
   - EMPLOYEE.NAME
   - EMPLOYEE.REMAINDERDAYS
   - EMPLOYEE_HIERARCHY.PARENT_DNUMBER
   - EMPLOYEE_HIERARCHY.VALID_FROM_DATE
   - EMPLOYEE_HIERARCHY.VALID_TO_DATE

6. Connect the *Join* node to *Aggregation* node.

7. In the *Aggregation* node, add the following columns:

Column Name	Action
DNUMBER	Add To Output
NAME	Add To Output
REMAINDERDAYS	Add As Aggregated Column
PARENT_DNUMBER	Add To Output
VALID_FROM_DATE	Add To Output
VALID_TO_DATE	Add To Output

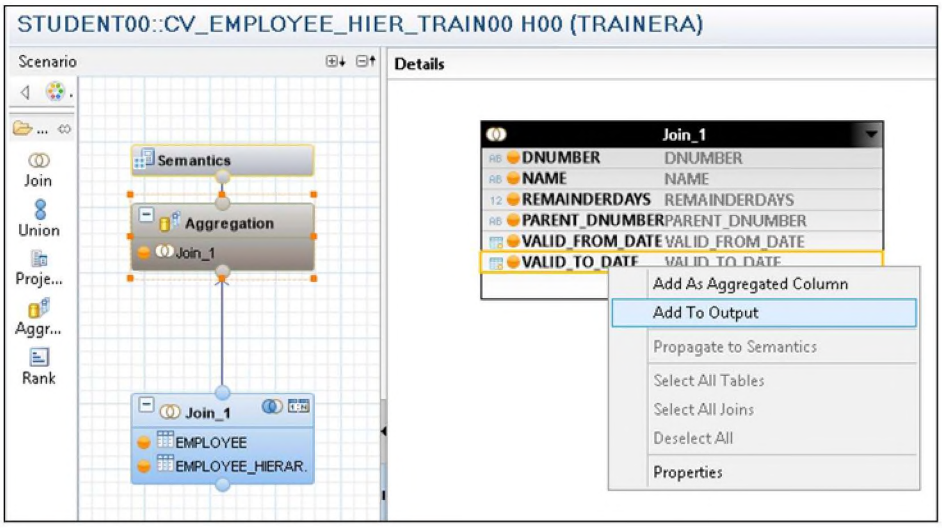


Figure 100: Aggregation Node — Add to Output

8. In the *Semantics* node, ensure that the *REMAINDERDAYS* column is assigned as a measure, and assign the remaining columns as attributes.

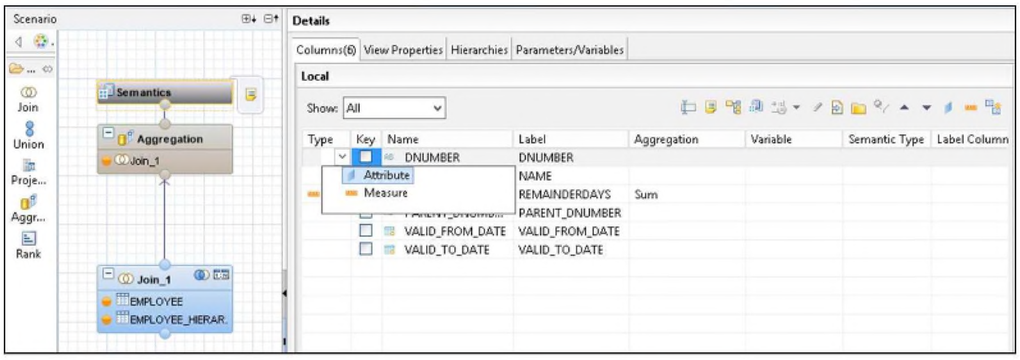


Figure 101: Assign Column Types

9. In the *Hierarchy* tab of the *Semantics* node, create a new hierarchy with the following properties:

Field	Value
Name	EMPLOYEE_HIERARCHY
Label	Employee Parent-Child Hierarchy
Hierarchy Type	Parent-Child Hierarchy
Child	DNUMBER
Parent	PARENT_DNUMBER

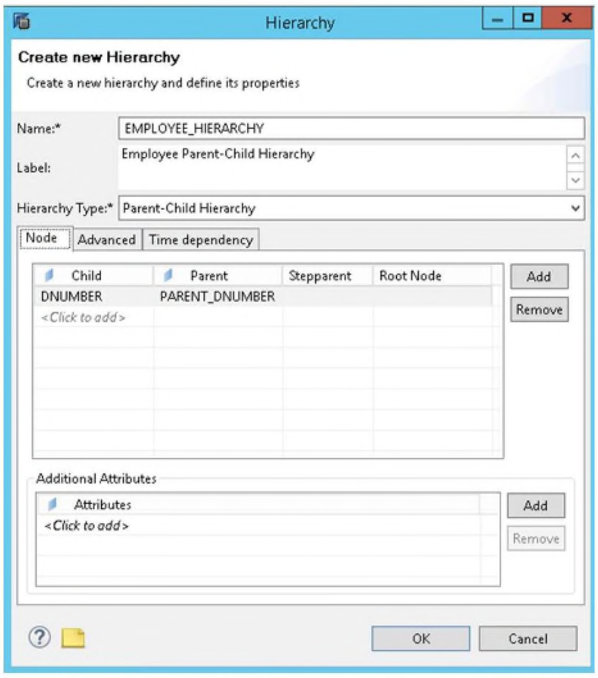


Figure 102: Parent-Child Hierarchy Properties

10. In the *Columns* tab of the *Semantics* node, assign the *NAME* attribute as the *Label Column* of *DNUMBER* row.

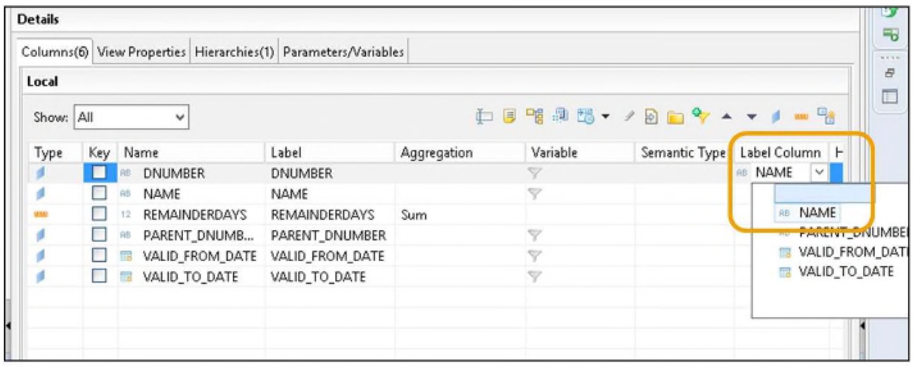


Figure 103: Label Column

11. Save and Activate the calculation view.

> **Note:**
> If you preview the data of the calculation view, you will not see the hierarchy that you have created, as SAP HANA Studio does not support hierarchies in preview mode.

Unit 2
Solution 8

Implement a Parent-Child Hierarchy

Business Example

You have been asked to build an Employee hierarchy.

The source data is available as a parent-child hierarchy.

> Note:
> In this exercise, when values include ##, replace the characters with the number your instructor provided to you.

1. Create a new calculation view using the data in the following table:

Field	Value
Name	**CV_EMPLOYEE_HIER_TRAIN##**
Label	**Employee Hierarchy**
Package	*STUDENT##*
View Type	*Calculation View*
Subtype	*Standard*
Type	*Graphical*
Data Category	*CUBE*

   a) Select *New* → *Calculation View* from the context menu of your package in the *Content* node.

   b) Enter the view properties as shown in the table.

   c) Choose *Finish*.

2. In the calculation view scenario, under the *Aggregation* node, add a *Join* node.

   a) In the *Scenario* Palette, choose the *Join* node.

   b) In the *Scenario* node, place a new *Join* node under the *Aggregation* node.

3. Add the following tables to the *Join* node:

   - TRAINING.EMPLOYEE
   - TRAINING.EMPLOYEE_HIERARCHY

   a) Drag the above tables to the *Join* node from the TRAINING schema of the *Catalog* node.

4. In your *Join* node, create a join between the tables:

Field	Value
Left Table	TRAINING.EMPLOYEE
Right Table	TRAINING.EMPLOYEE_HIERARCHY
Join Columns	EMPLOYEE.DNUMBER = EMPLOYEE_HI-ERARCHY.CHILD_DNUMBER
Join Type	Left Outer
Cardinality	1..n

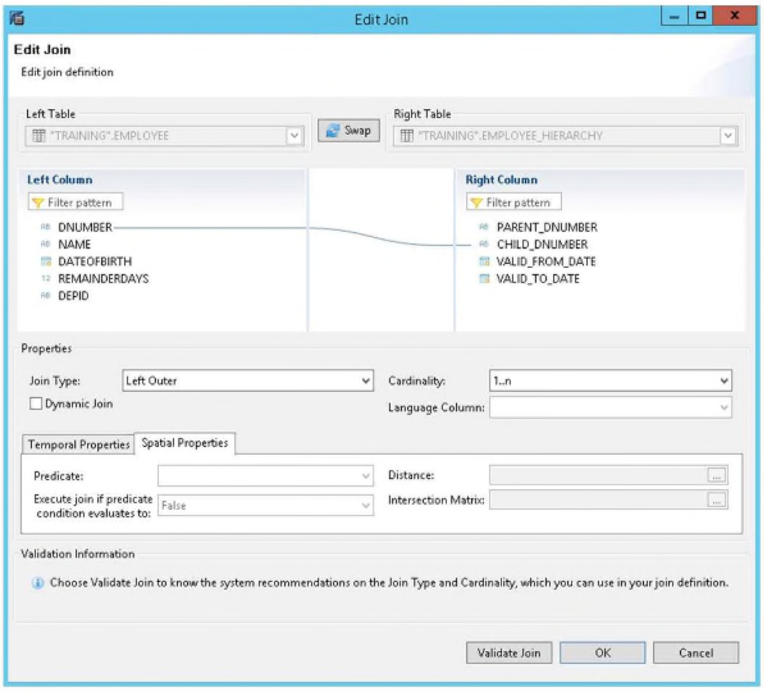


Figure 99: Join of EMPLOYEE and EMPLOYEE_HIERARCHY

a) Drag a connecting line between the columns EMPLOYEE.DNUMBER and EMPLOYEE_HIERARCHY.CHILD_DNUMBER.

b) Double-click the join connector and define the properties as shown in the table.

5. In the *Join* node, add the following columns to output:

- EMPLOYEE.DNUMBER
- EMPLOYEE.NAME
- EMPLOYEE.REMAINDERDAYS
- EMPLOYEE_HIERARCHY.PARENT_DNUMBER
- EMPLOYEE_HIERARCHY.VALID_FROM_DATE
- EMPLOYEE_HIERARCHY.VALID_TO_DATE

a) In the *Join* node, right-click the above columns and select *Add To Output* from the context menu.

6. Connect the *Join* node to *Aggregation* node.

a) In the Scenario of the calculation view, drag a connecting line from the join node to the aggregation node.

7. In the *Aggregation* node, add the following columns:

Column Name	Action
DNUMBER	Add To Output
NAME	Add To Output
REMAINDERDAYS	Add As Aggregated Column
PARENT_DNUMBER	Add To Output
VALID_FROM_DATE	Add To Output
VALID_TO_DATE	Add To Output

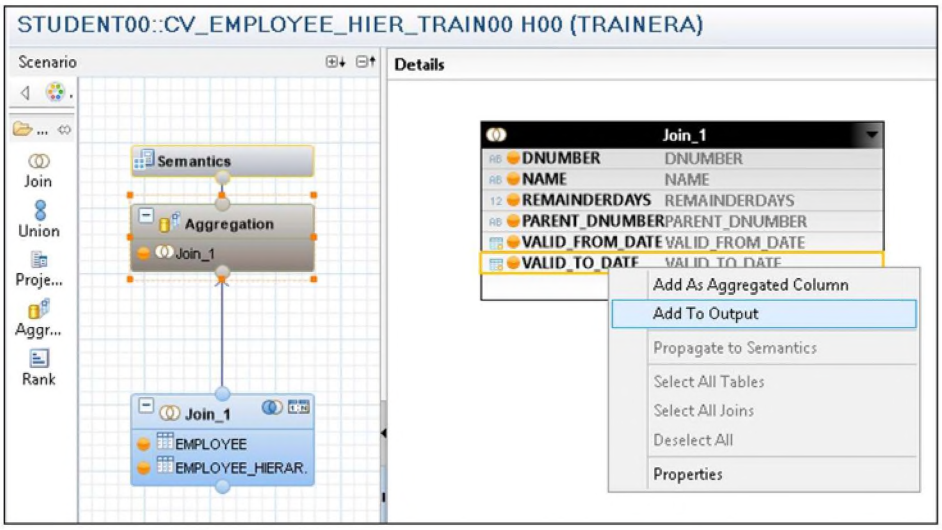


Figure 100: Aggregation Node — Add to Output

a) In the *Aggregation* node, right-click each column and select *Add To Output/Add As Aggregated Column* as shown in the table

8. In the *Semantics* node, ensure that the *REMAINDERDAYS* column is assigned as a measure, and assign the remaining columns as attributes.

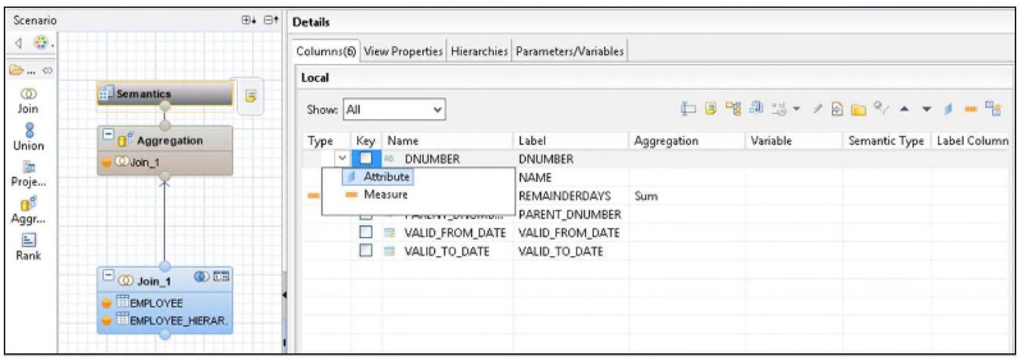


Figure 101: Assign Column Types

a) In the *Columns* tab of the *Semantics* node, ensure that the *REMAINDERDAYS* column is assigned as Type: *Measure*.

b) Assign the remaining columns as Type: *Attribute*

9. In the *Hierarchy* tab of the *Semantics* node, create a new hierarchy with the following properties:

Field	Value
Name	EMPLOYEE_HIERARCHY
Label	Employee Parent-Child Hierarchy
Hierarchy Type	Parent-Child Hierarchy
Child	DNUMBER
Parent	PARENT_DNUMBER

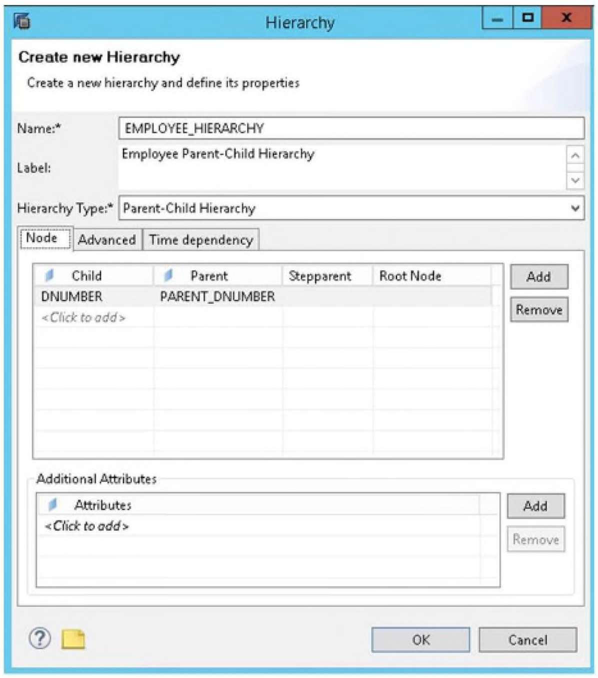


Figure 102: Parent-Child Hierarchy Properties

a) Select the *Hierarchies* tab of the *Semantics* node.

b) Choose the + button to create a new hierarchy.

c) Enter the *Name* and *Label* as shown in the table.

d) In the *Child* column of the *Node* tab, choose the text *<Click to Add>* and choose the *DNUMBER* attribute.

e) In the *Parent* column of the *Node* tab, choose the text *Select Parent* and choose the *PARENT_DNUMBER* attribute.

f) Choose *OK*.

10. In the *Columns* tab of the *Semantics* node, assign the *NAME* attribute as the *Label Column* of *DNUMBER* row.

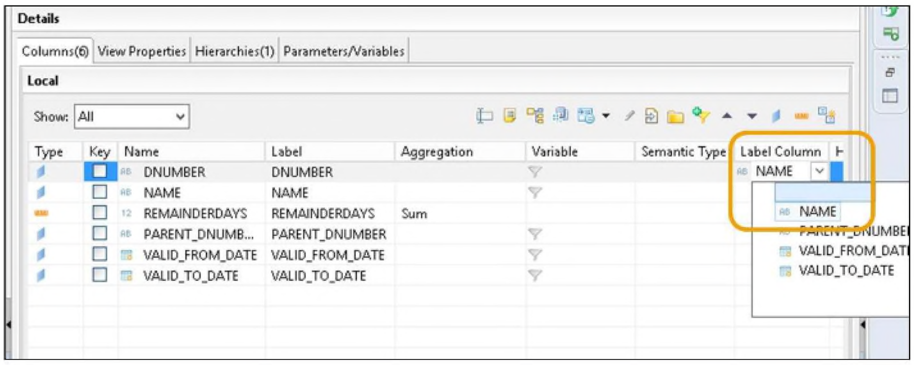


Figure 103: Label Column

a) In the *DNUMBER* row, select the *NAME* attribute from the drop-down menu of *Label Column*.

11. Save and Activate the calculation view.

> Note:
> If you preview the data of the calculation view, you will not see the hierarchy that you have created, as SAP HANA Studio does not support hierarchies in preview mode.

Unit 2
Exercise 9

Implement a Time-Dependent Hierarchy

Business Example

You have been asked to build a Calculation view to analyze Cost Center expenses.

The source data is time dependent and is stored as a parent-child hierarchy.

Since the source data is time dependent, you will need the view to allow for a selection of an effective date for the hierarchy.

> Note:
> In this exercise, when values include ##, replace the characters with the number your instructor provided to you.

Task 1: Create a Dimension Calculation View with a Time-Dependent Hierarchy

1. Create a new dimension calculation view using the data in the following table:

Field	Value
Name	CV_CCTR_HIER_##
Label	Cost Center with Hierarchy
Package	STUDENT##
View Type	Calculation View
Subtype	Standard
Type	Graphical
Data Category	DIMENSION

2. Add the following table to the *Projection* node:
   - TRAINING.CCTR_HIE_TIME
3. In the *Projection* node, add all columns to output.
4. In the *Semantics* node, define column CCTR_CHILD as the Key column.
5. In the *Parameters/Variables* tab of the *Semantics* node, select the *Create Input Parameter* from the + drop-down menu, and use the following details:

Field	Value
Name	EFFECTIVE_DATE
Label	Hierarchy Effective Date
Parameter Type	Direct

Field	Value
Default Value → Type	Constant
Default Value → Value	2015–01–01
Semantic Type	Date
Data Type	DATE

> Note:
> The Input Parameter will be used to select the effective date of the hierarchy.

6. In the *Hierarchy* tab of the *Semantics* node, create a new Hierarchy using the following details:

Field	Value
Name	HIER_CCTR_TIME_DEPENDENT
Label	Time Dependent Cost Center Hierarchy
Hierarchy Type	Parent-Child Hierarchy
Child Attribute	CCTR_CHILD
Parent Attribute	CCTR_PARENT

7. Enable time dependency for the hierarchy using the details in the following table:

Field	Value
Enable time dependency	Yes
Valid From Column	DATE_FROM
Valid To Column	DATE_TO
Key Date → Key Parameter	*Input Parameters → EFFECTIVE_DATE*

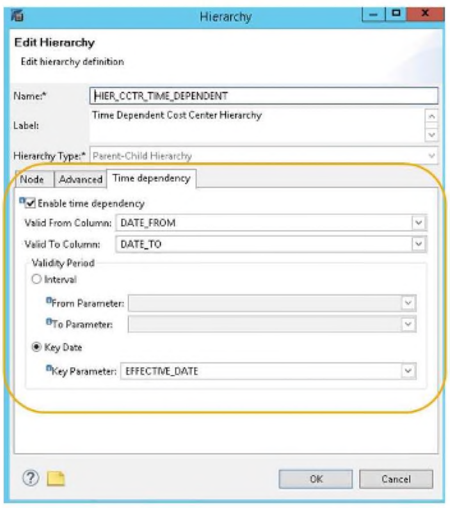


Figure 104: Time Dependency Properties

8. Save and activate the view.

Task 2: Create a Calculation View Utilizing the Time-Dependent Cost Center Hierarchy
Create a Star Join-type Calculation View using Cost data, utilizing the created time-dependent Cost Center Hierarchy.

1. Create a new calculation view using the data in the following table:

Field	Value
Name	CV_CCTR_EXPENSES_##
Label	Cost Center Expenses
Package	STUDENT##
View Type	Calculation View
Subtype	Standard
Type	Graphical
Data Category	CUBE
With Star Join	Yes

Note:
Select the *With Star Join* property of the *New Information View* window.

2. In the *Calculation View* scenario, under the *Star Join*, add an *Aggregation* node.
3. Add the following table to the *Aggregation* node:
   - TRAINING.CCTR_COST
4. Connect the *Aggregation* node to *Star Join* node.
5. In the *Aggregation* node, add the following columns:

Column Name	Action
CCTR	Add To Output
COST	Add As Aggregated Column

6. Add the Dimension Calculation View *CV_CCTR_HIER_##* from your package to the *Star Join* node.
7. In the *Star Join* node, define a join between the *Aggregation_1* node and the *CV_CCTR_HIER_##* view using the properties in the following table:

Field	Value
Left Column:	Aggregation_1.CCTR
Right Column:	STUDENT##::CV_CCTR_HI-ER_##.CCTR_CHILD
Join Type:	Referential

Field	Value
Cardinality:	1..n

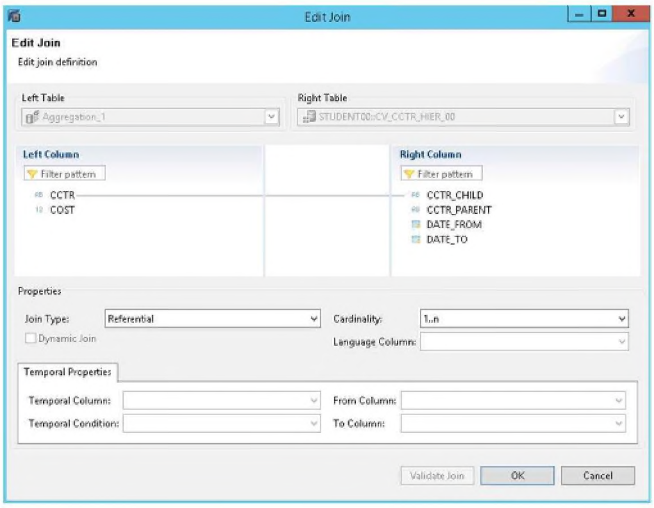


Figure 105: Star Join — Edit Join

8. In the *Star Join* node, add column *Aggregation_1.COST* to output.
9. In the *Semantics* node, define the COST column as a measure.
10. *Manage Mappings* of Input Parameters, and select *Copy and Map 1:1* of the EFFECTIVE_DATE Input Parameter from the *STUDENT##:CV_CCTR_HIER_##* Dimension Calculation View.

> Note:
> To Manage Mappings, choose *Input Parameter Manage Mapping* in the *Parameters/Variables* tab of the *Semantics* node

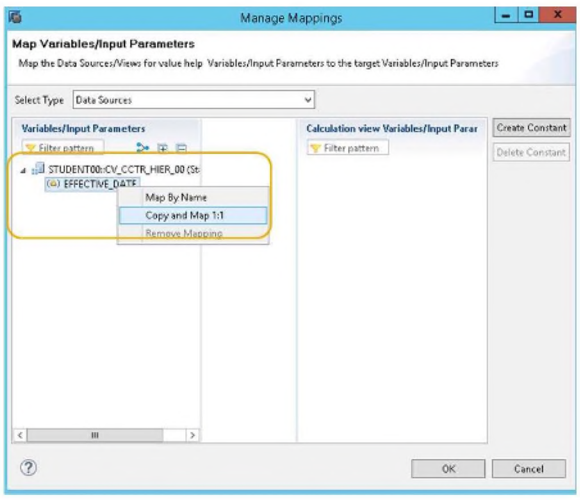


Figure 106: Manage Mappings

11. Save and activate the view.

Task 3: Query the time-dependent hierarchy using Analysis for Microsoft Excel

1. Start Analysis for Microsoft Excel.

2. In the *ANALYSIS* tab of Microsoft Excel, select *Insert Data Source* and choose *Select Data Source....*
3. In the *Logon to SAP BusinessObjects BI Platform* screen, choose *Skip*.
4. Right-click in the blank area of the *Select Data Source* screen, and select *Create New SAP HANA Connection*.
5. Enter the details as shown in the table, and choose *Save*.

> Note:
> Remember to replace ## with your assigned student number.

Field	Value
Description	**HANA_##**
Scheme	HTTP
Host Name	**wdflbmt7215.wdf.sap.corp**
Port	**8000**
Authentication Type	*Basic*

6. Select your HANA_## connection, choose *Next* and logon using your *STUDENT##* user.
7. In the *Area* tab, choose *STUDENT## → Cost Center Expenses* and choose *OK*.
8. In the *Prompts for Cost Center Expenses* window, leave the *Hierarchy Effective Date* prompt as per default (2015–01–01) and choose *OK*.
9. Add the *Time Dependent Cost Center hierarchy* of the *CCTR_CHILD → Hierarchies* node to the ROWS area.
10. Expand the *CCTR_CHILD* hierarchy to Level 02.

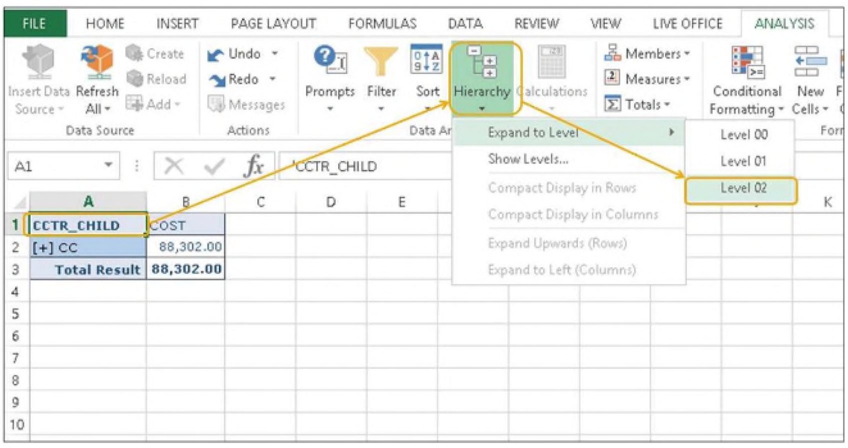


Figure 109: Expand Hierarchy to Level 02

11. You are now viewing the hierarchy as of 2015–01–01. Take a note of under which node in the hierarchy you find cost center *CC21*.
12. Refresh the query using the Effective Date value: **2016–02–01**.

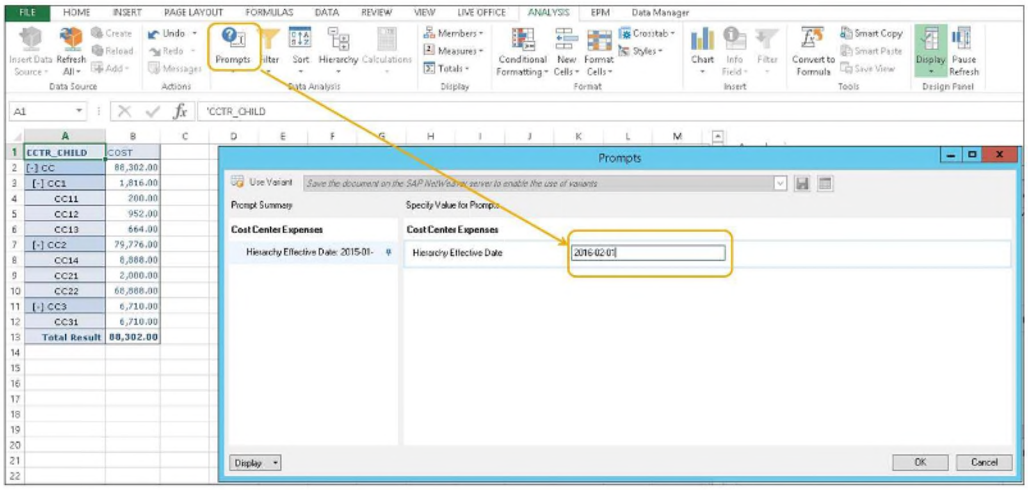


Figure 110: Refresh Query with new Effective Date

13. Note that the cost center *CC21* now has moved in the hierarchy.

Implement a Time-Dependent Hierarchy

Business Example

You have been asked to build a Calculation view to analyze Cost Center expenses.

The source data is time dependent and is stored as a parent-child hierarchy.

Since the source data is time dependent, you will need the view to allow for a selection of an effective date for the hierarchy.

> Note:
> In this exercise, when values include ##, replace the characters with the number your instructor provided to you.

Task 1: Create a Dimension Calculation View with a Time-Dependent Hierarchy

1. Create a new dimension calculation view using the data in the following table:

Field	Value
Name	CV_CCTR_HIER_##
Label	Cost Center with Hierarchy
Package	STUDENT##
View Type	Calculation View
Subtype	Standard
Type	Graphical
Data Category	DIMENSION

   a) Select *New* → *Calculation View* from the context menu of your package in the *Content* node.

   b) Enter the view properties as per above table.

   c) Choose *Finish*.

2. Add the following table to the *Projection* node:

   - TRAINING.CCTR_HIE_TIME

   a) Drag the above table to the *Projection* node from the TRAINING schema of the *Catalog* node.

3. In the *Projection* node, add all columns to output.

a) In the *Details of the Projection* node, right-click the TRAINING.CCTR_HIE_TIME table and select *Add All To Output* from the context menu.

4. In the *Semantics* node, define column CCTR_CHILD as the Key column.

   a) In the *Columns* tab of the *Semantics* node, select the *Key* property for the *CCTR_CHILD* row.

5. In the *Parameters/Variables* tab of the *Semantics* node, select the *Create Input Parameter* from the + drop-down menu, and use the following details:

Field	Value
Name	EFFECTIVE_DATE
Label	Hierarchy Effective Date
Parameter Type	Direct
Default Value → Type	Constant
Default Value → Value	2015–01–01
Semantic Type	Date
Data Type	DATE

Note:
The Input Parameter will be used to select the effective date of the hierarchy.

6. In the *Hierarchy* tab of the *Semantics* node, create a new Hierarchy using the following details:

Field	Value
Name	HIER_CCTR_TIME_DEPENDENT
Label	Time Dependent Cost Center Hierarchy
Hierarchy Type	Parent-Child Hierarchy
Child Attribute	CCTR_CHILD
Parent Attribute	CCTR_PARENT

   a) Choose + in the *Hierarchy* tab of the *Semantics* node and define the properties using the details in the above table.

7. Enable time dependency for the hierarchy using the details in the following table:

Field	Value
Enable time dependency	Yes
Valid From Column	DATE_FROM
Valid To Column	DATE_TO

Field	Value
Key Date → Key Parameter	*Input Parameters → EFFECTIVE_DATE*

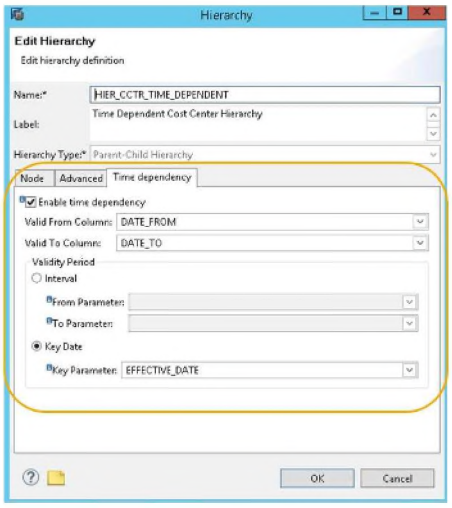


Figure 104: Time Dependency Properties

a) Select the *Time dependency* tab of the *Edit Hierarchy* screen and enter the properties as shown in the table.

b) Choose *OK*

8. Save and activate the view.

   a) Choose *Save and Activate*.

Task 2: Create a Calculation View Utilizing the Time-Dependent Cost Center Hierarchy
Create a Star Join-type Calculation View using Cost data, utilizing the created time-dependent Cost Center Hierarchy.

1. Create a new calculation view using the data in the following table:

Field	Value
Name	CV_CCTR_EXPENSES_##
Label	Cost Center Expenses
Package	STUDENT##
View Type	Calculation View
Subtype	Standard
Type	Graphical
Data Category	CUBE
With Star Join	Yes

Note:
Select the *With Star Join* property of the *New Information View* window.

a) Select *New* → *Calculation View* from the context menu of your package in the *Content* node.

b) Enter the view properties as shown in the table.

c) Choose *Finish*.

2. In the *Calculation View* scenario, under the *Star Join*, add an *Aggregation* node.

a) In the *Scenario* Palette, choose the *Aggregation* node.

b) In the *Scenario*, place a new *Aggregation* node under the *Star join* node.

3. Add the following table to the *Aggregation* node:

- TRAINING.CCTR_COST

a) Drag the above tables to the *Join* node from the TRAINING schema of the *Catalog* node.

4. Connect the *Aggregation* node to *Star Join* node.

a) In the Scenario of the calculation view, drag a connecting line from the *Aggregation* node to the *Star Join* node.

5. In the *Aggregation* node, add the following columns:

Column Name	Action
CCTR	Add To Output
COST	Add As Aggregated Column

a) In the *Aggregation* node, right-click each column and select *Add To Output/Add As Aggregated Column* as shown in the table.

6. Add the Dimension Calculation View *CV_CCTR_HIER_##* from your package to the *Star Join* node.

a) From your package in the Systems view, drag the Calculation View *CV_CCTR_HIER_##* to the *Details* pane of the *Star Join* node.

7. In the *Star Join* node, define a join between the *Aggregation_1* node and the *CV_CCTR_HIER_##* view using the properties in the following table:

Field	Value
Left Column:	Aggregation_1.CCTR
Right Column:	STUDENT##::CV_CCTR_HI-ER_##.CCTR_CHILD
Join Type:	Referential
Cardinality:	1..n

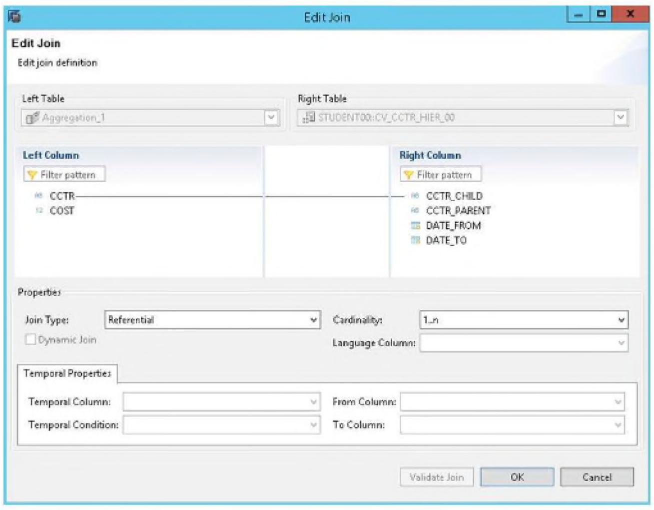


Figure 105: Star Join — Edit Join

8. In the *Star Join* node, add column *Aggregation_1.COST* to output.
   a) In the details of the *Star Join* node, right-click column *Aggregation_1.COST* and select *Add To Output* from the context menu.
9. In the *Semantics* node, define the COST column as a measure.
   a) In the *Semantics* node, select the *Columns* tab of the *Details* pane.
   b) Define the *Type* of the COST row as *Measure*.
10. *Manage Mappings* of Input Parameters, and select *Copy and Map 1:1* of the EFFECTIVE_DATE Input Parameter from the *STUDENT##:CV_CCTR_HIER_##* Dimension Calculation View.

> Note:
> To Manage Mappings, choose *Input Parameter Manage Mapping* in the *Parameters/Variables* tab of the *Semantics* node

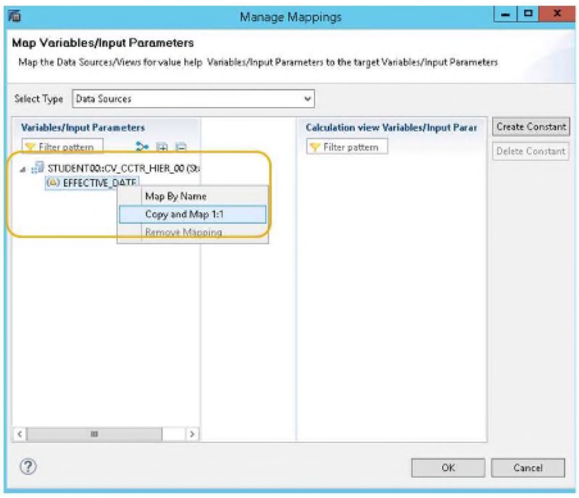


Figure 106: Manage Mappings

   a) Select the *Parameters/Variables* tab of the *Semantics* node.
   b) Choose *Input Parameter Manage Mapping*.

c) Right-click the *EFFECTIVE_DATE* input parameter and select *Copy and Map 1:1* from the context menu.

11. Save and activate the view.

a) Choose *Save and Activate*.

Task 3: Query the time-dependent hierarchy using Analysis for Microsoft Excel

1. Start Analysis for Microsoft Excel.

a) Using the *Start Menu*, search for **`Analysis for Microsoft Excel`** and start the application.

b) Select a blank worksheet.

2. In the *ANALYSIS* tab of Microsoft Excel, select *Insert Data Source* and choose *Select Data Source...*.

a) Select the *ANALYSIS* tab.

b) Select *Insert Data Source*.

c) Select *Select Data Source...*

3. In the *Logon to SAP BusinessObjects BI Platform* screen, choose *Skip*.

4. Right-click in the blank area of the *Select Data Source* screen, and select *Create New SAP HANA Connection*.

a) Create a new SAP HANA Connection.

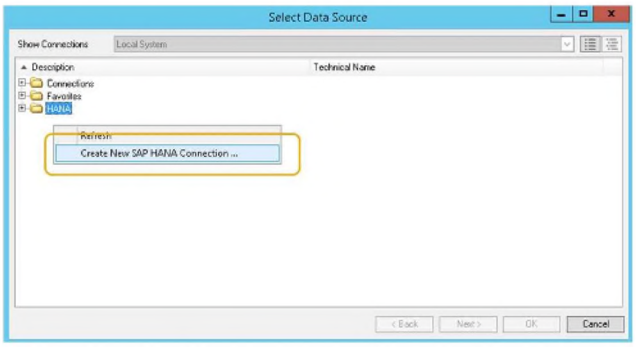


Figure 107: Create New SAP HANA Connection

5. Enter the details as shown in the table, and choose *Save*.

> Note:
> Remember to replace ## with your assigned student number.

Field	Value
Description	**`HANA_##`**
Scheme	`HTTP`
Host Name	**`wdflbmt7215.wdf.sap.corp`**
Port	**`8000`**

Field	Value
Authentication Type	*Basic*

6. Select your HANA_## connection, choose *Next* and logon using your *STUDENT##* user.
7. In the *Area* tab, choose *STUDENT##* → *Cost Center Expenses* and choose *OK*.
8. In the *Prompts for Cost Center Expenses* window, leave the *Hierarchy Effective Date* prompt as per default (2015–01–01) and choose *OK*.
9. Add the *Time Dependent Cost Center hierarchy* of the *CCTR_CHILD* → *Hierarchies* node to the ROWS area.

   a) Add the hierarchy to the *ROWS* pane, as shown in the figure.

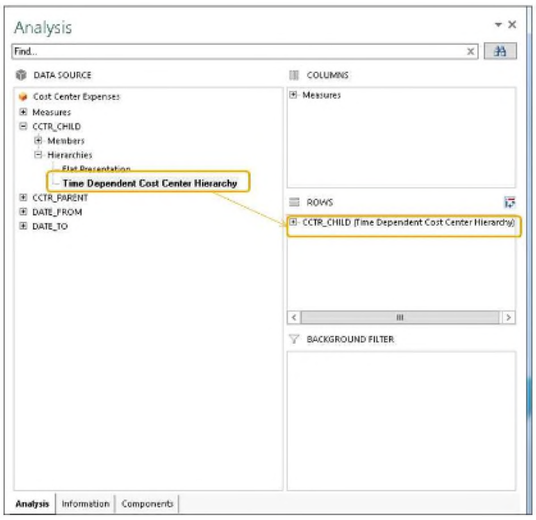


Figure 108: Add Hierarchy to Rows

10. Expand the *CCTR_CHILD* hierarchy to Level 02.

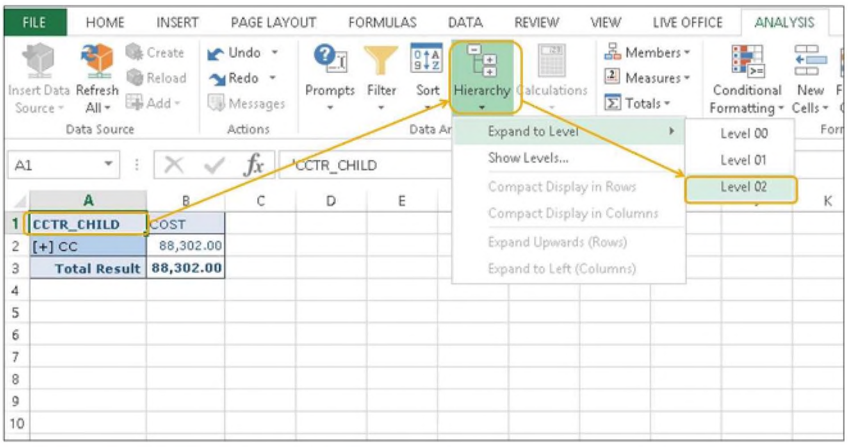


Figure 109: Expand Hierarchy to Level 02

   a) Select the *CCTR_CHILD* cell (A1).

   b) Select *Hierarchy* → *Expand to Level* → *Level 02*.

11. You are now viewing the hierarchy as of 2015–01–01. Take a note of under which node in the hierarchy you find cost center *CC21*.
12. Refresh the query using the Effective Date value: **2016–02–01**.

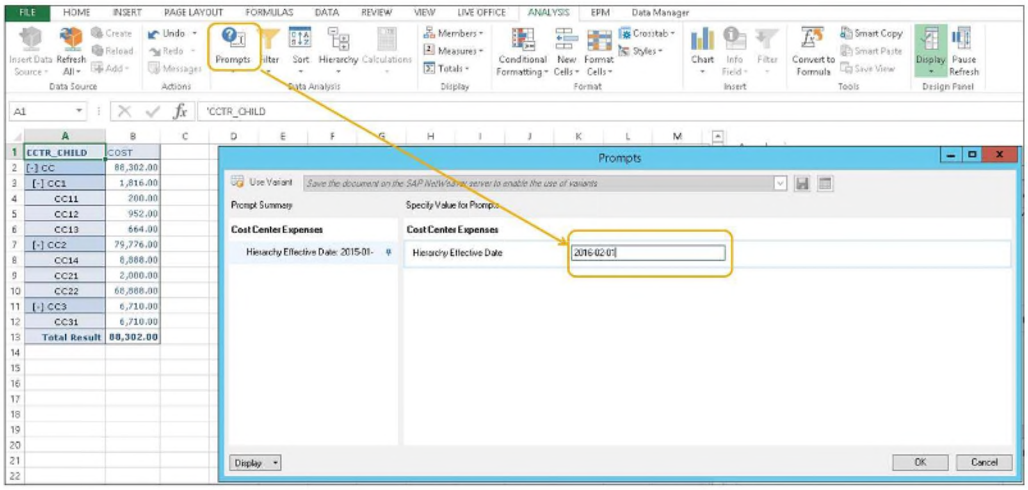


Figure 110: Refresh Query with new Effective Date

a) In the *Analysis* tab, choose *Prompts*.

b) Enter Hierarchy Effective Date: **2016–02–01**.

c) Choose *OK*.

13. Note that the cost center *CC21* now has moved in the hierarchy.

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

LESSON SUMMARY

You should now be able to:

- Use hierarchies
- Create parent-child hierarchies
- Work with time-based hierarchies

Unit 2
Lesson 2

Creating Restricted and Calculated Columns

LESSON OVERVIEW

This lesson explains how to use different modeling capabilities on columns. In particular, you will learn how to create calculated columns (attributes or measures) and restricted columns (for attribute only).

Business Example

You want to ensure you minimize the processing in the client tool, pushing down expressions to SAP HANA using calculated and restricted columns.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create restricted and calculated columns
- Create Restricted Columns
- Create calculated columns

Restricted Columns

What is a restricted column?

- As the name implies it is a column that does not give the complete picture of a column, it is restricted to a **subset** of the original column.
- The benefit of a restricted column is that it expands the modeling options in a view, giving the modeler the possibilities of creating objects that can be easily reported on or reused.

Country	Month	Amount	US Amount
DE	2010-01	12.345,00	0
DE	2010-02	15.678,00	0
DE	2010-03	25.814,00	0
DE	2010-04	21.586,00	0
DE	2010-05	21.861,00	0
DE	2010-06	11.258,00	[illegible]
DE	2010-07	12.387,00	0
DE	2010-08	13.589,00	0
DE	2010-09	12.345,00	0
DE	2010-10	15.678,00	0
DE	2010-11	25.814,00	0
DE	2010-12	21.586,00	0
US	2010-01	21.861,00	21.861,00
US	2010-02	11.258,00	11.258,00
US	2010-03	12.387,00	12.387,00
US	2010-04	13.589,00	13.589,00
US	2010-05	12.345,00	12.345,00
US	2010-06	15.678,00	15.678,00
US	2010-07	25.814,00	25.814,00
US	2010-08	21.586,00	21.586,00
US	2010-09	21.861,00	21.861,00
US	2010-10	11.258,00	11.258,00
US	2010-11	12.387,00	12.387,00
US	2010-12	25.814,00	25.814,00

Restricted column based on **Amount**, where **Country** = **US**

Figure 111: The Benefits of Restricted Columns

The restricted column is one of a set of columns available in SAP HANA models, which includes the following:

- Columns
- Calculated columns
- Restricted columns

The restricted column is restricted based on one or more attributes. These columns can be anything in the base table or view that the modeler defines to help reporting or further modeling.

> Note:
> The restriction cannot be based on a column defined as a *Measure* in the semantics.

Example Without Using a Restricted Column

- You have a transactional table with cost data items, with each cost type split on a different line.
- If you want to find out the shipping cost you could create an analytic view with Cost Type as an attribute, and Amount as a measure.
- You could then restrict your report by reporting on Cost Type, setting the column filtered on Cost Type = "Shipping Cost" in the reporting tool.

Cost Type	Amount
Purchasing Price	€ 1 200
Shipping Cost	€ 80
VAT	€ 346
Processing Cost	€ 150
Margin	€ 300

Figure 112: Example Without Using a Restricted Column

With this data, you could restrict your report by filtering on Cost Type = Shipping Cost. If your reporting tool is SAP Business Objects, you could create a data provider with a query restriction where Cost Type is filtered to only display Shipping Cost.

Creating Restricted Columns

- A restricted column can be created in an aggregation node of a calculation view.
- You assign a measure to the restricted column, and also which columns define the restriction. For example, we can create restricted column based on Gross Amount restricted to certain product categories.

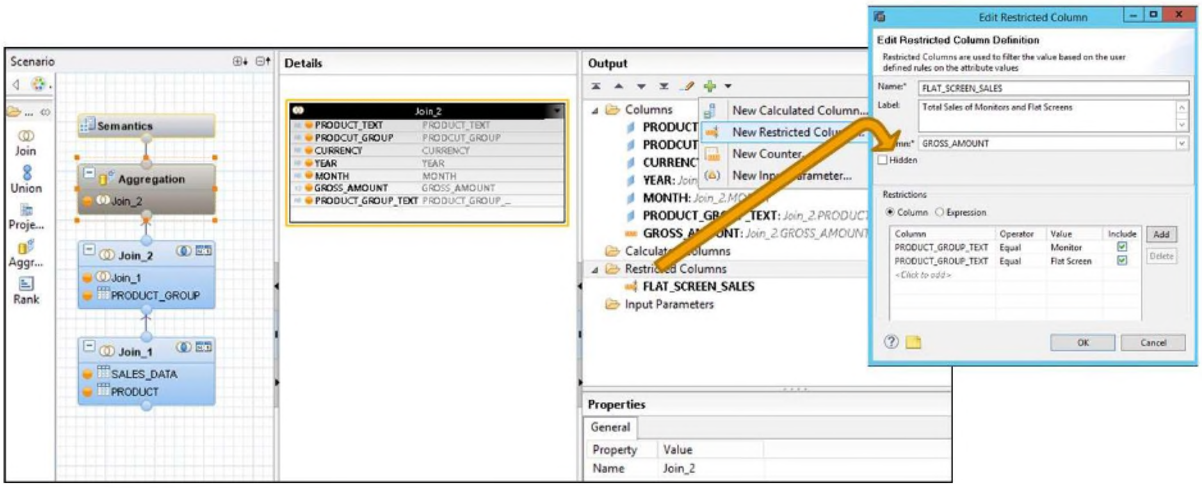


Figure 113: Creating Restricted Columns

Setting the Restriction

Continuing with the example of using SAP Business Objects for reporting, when you have access to this restricted column, you can report on both the total gross amount and the flat screens gross amount in the same data provider or query.

- The restrictions for a restricted column do not have to be limited to one single Column: you can filter based on multiple columns depending on your reporting requirements.
- Multiple operators can be used for the filter restrictions, such as Equal, Greater Than, Less Than and others.
- You have the option to hide the restricted column, for example to reuse it in a calculated column and thereby make it unavailable for reporting.

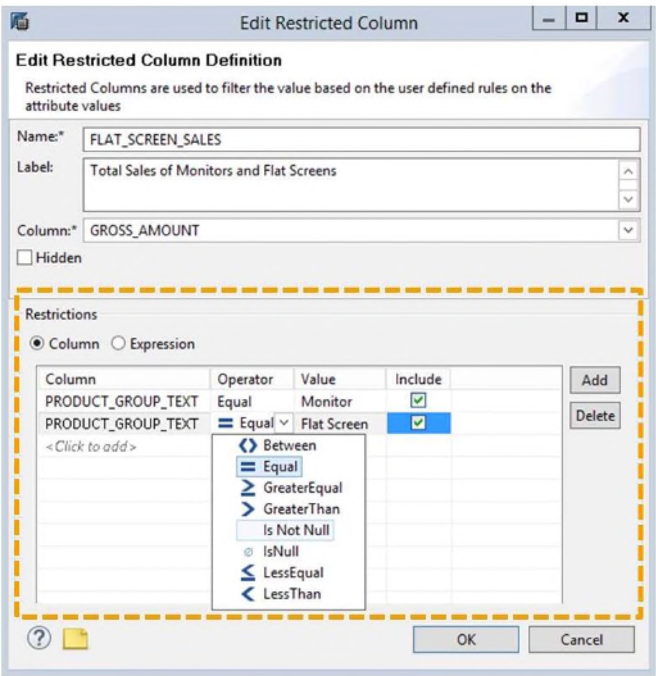


Figure 114: Setting the Restriction

If there are different lines in the restriction, all the lines defined on the same column are combined with the logical operator *OR*, and then, all the sets of restrictions for different columns are combined with the logical operator *AND*.

> Note:
> This is the case regardless of the order in which you define the lines.

From SAP HANA 1.0 SPS09, in Calculation Views, it is possible to create restricted columns with the *Between* operator, and using input parameters.

Displaying and Editing Restriction Expression

- Several restrictions on the same column are combined with an ***OR*** operator.
- Restrictions on different columns are combined with an ***AND*** operator.
- To visualize the expression, select the *Columns* tab.
- You can also edit the expression to enrich it.

  In this case, the column-based definition can no longer be edited.

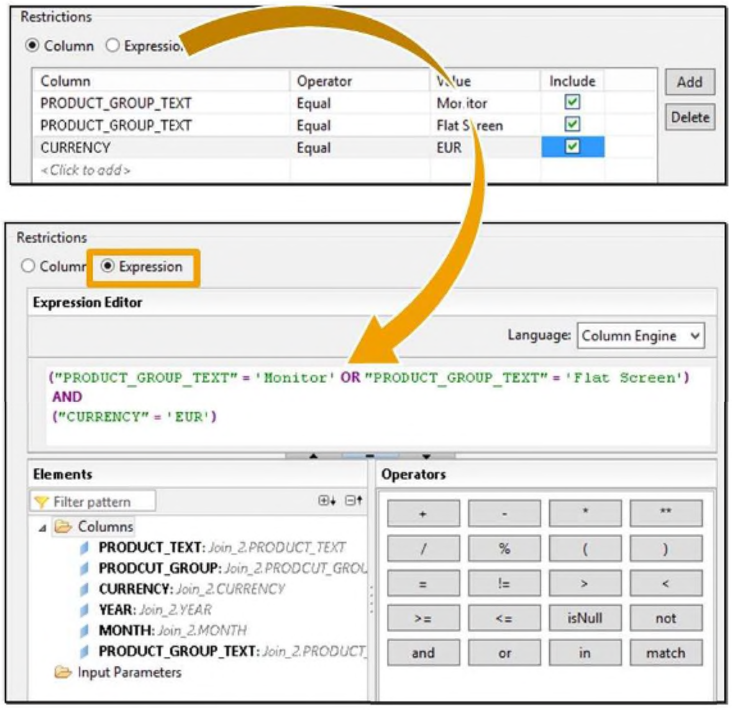


Figure 115: Displaying and Editing Restriction Expression

You can visualize the expression corresponding to your restrictions. This can be useful for example when you want to check the precedence of logical operators.

It is also possible to modify this expression, especially in complex scenarios when the features offered in the *Column* tab do not fulfill your requirements.

From SPS11 onwards, the expression can be written using the SQL language, in addition to the Column Engine syntax. However, for expressions in SQL, SAP HANA modeling supports only a limited list of SQL functions.

> Note:
> The general SQL support within expressions is not specific to restricted measure expressions. The SQL language can also be used in the following expressions:
>
> - Calculated Columns (since SAP HANA SPS10)
> - Filters (new in SPS11)
> - Default values for Variables and Input Parameters (new in SPS11)
>
> You will learn about these different types of modeling functions later.

Calculated Columns

- In a data model sometimes not all measures available in the base data will give your users sufficient information for reporting if you just provide the base measures in your views.
- In order to enhance the data, SAP HANA provides Calculated Columns where the modeler is able to include calculations already within the information model in order to help reporting or further modeling.

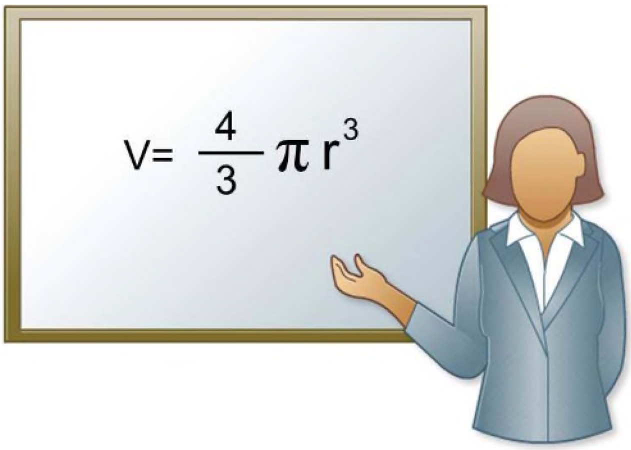


Figure 116: When to Use Calculated Columns

A calculated column does not have to be a complex formula, it can also be a simple calculation.

When to use Calculated Columns

- When you include calculations in your views using calculated columns you take advantage of the speed of SAP HANA letting the database engine perform calculations instead of doing these calculations in your end client reporting tool.
- Having ready defined calculations in information models can also help simplifying reporting by unifying calculations.
- ***HINT:*** Only create calculated columns where there is a specific reporting need, as data transfer will take place between the OLAP engine and Calculation engine slowing down execution.

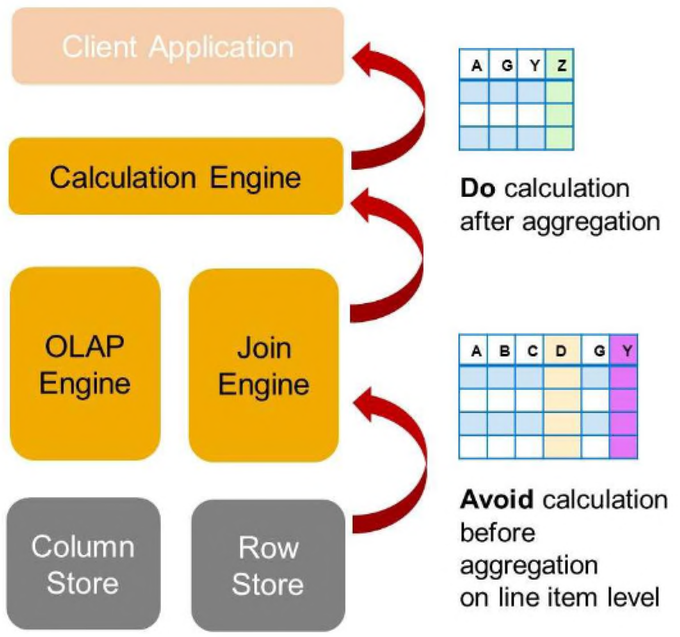
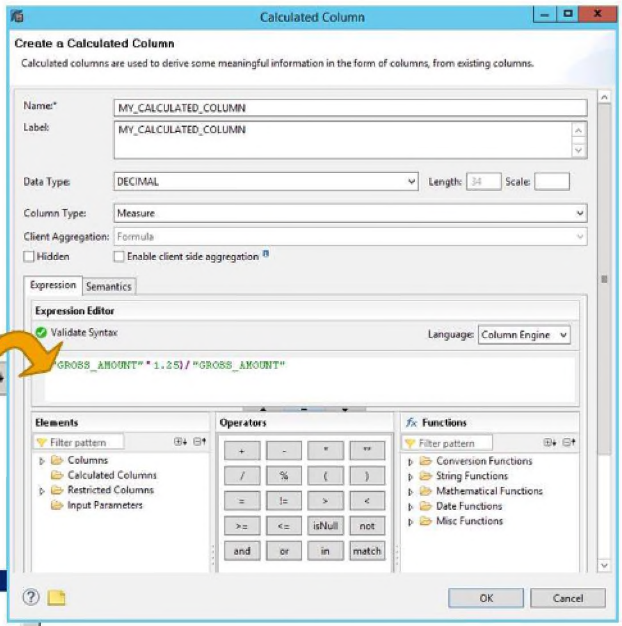


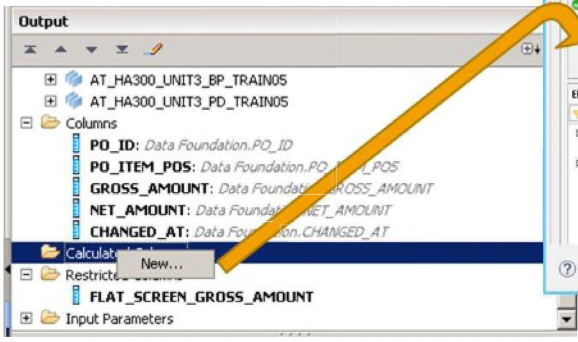
Figure 117: When to use Calculated Columns

The Calculated Columns Wizard

When creating a calculated column, the first step is to choose the Column Type, Measure, or Attribute.

- A calculated column is defined within an information model and can use the string functions, mathematical functions etc. available in the editor.
- You can define a calculated column as measure or attribute.





- Double-click or drag-and-drop Elements, Operators and Functions to build the expression
- When you type, use Ctrl+Space to autocomplete the name of functions, columns, input parameters.

Figure 118: Calculated Columns Wizard

Consider Granularity when Creating Calculated Columns

Product	Units	Price	Total Sales (Units * Price)
Keyboards	100	€ 40	€ 4 000
LCD Screens	50	€ 300	€ 15 000
Network Switches	75	€ 90	€ 6 750
VOIP Telephones	200	€ 65	€ 13 000
Servers	30	€ 2 000	€ 60 000
SUM:	~~455~~	~~€ 2 495~~	~~€ 1 135 225~~

- For certain measure it is not possible to perform the calculations when the measures are already aggregated. The aggregated granularity of, for example, Price does not mean anything.

Figure 119: Consider Granularity when Creating Calculated Columns

In the figure, Consider Granularity when Creating Calculated Columns, in the sum line highlighted red, the units have been aggregated as well as the price. Multiplying these to aggregates will not give a meaningful result.

Triggering Calculations at the Right Level

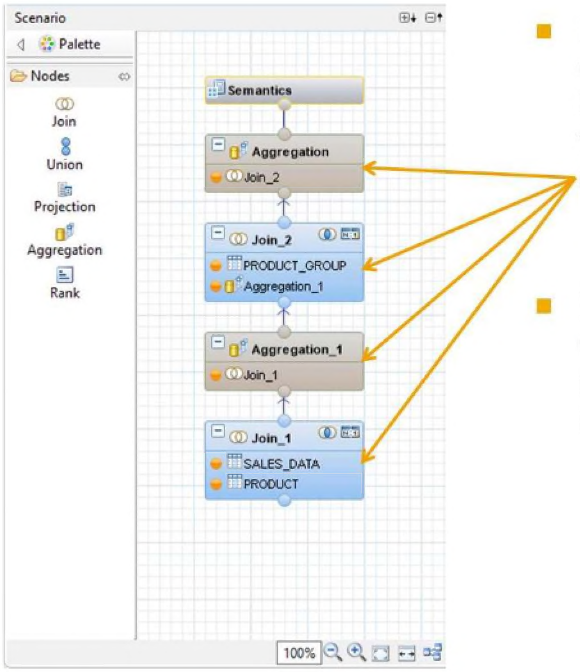


Figure 120: Triggering Calculations at the Right Level

By analyzing your reporting requirements, you can arrive at a decision at which stage the calculation should be performed.

> Caution:
> Try to minimize calculations before aggregation; for example, when calculations include multiplication or division.
>
> If the calculations are just additions or subtractions, it is not required. As well, it will slow down the execution of the view.

Calculation Before Aggregation

Product	Units	Price	Total Sales (Units * Price)
Keyboards	100	€ 40	€ 4 000
LCD Screens	50	€ 300	€ 15 000
Network Switches	75	€ 90	€ 6 750
VOIP Telephones	200	€ 65	€ 13 000
Servers	30	€ 2 000	€ 60 000
		SUM:	€ 98 750

- This way you can be sure that you end up with a correct sum as the calculation are performed on the correct granular level.

Figure 121: Calculation Before Aggregation

When data is calculated at the correct level of granularity, the total sales measure is correctly calculated, that is, 98,750.

Calculated Columns and Persistence

Creating calculated columns in an SAP HANA information view does not mean the calculation is persisted. It is simply projected through the generated column view.

The calculated column is available only during the runtime of the model and can be displayed in a report, or consumed by another view.

By contrast, if you need to keep the result of a calculation over time, you can apply one of the following methods:

Storing the Value of a Calculated Column

- Create an ad-hoc SQL artifact that updates and inserts the calculated column into a table.

  You can either use a write-enabled procedure or a table function.

- Use the transformation and calculation features provided by an ETL tool, such as SAP Landscape Transformation (SLT) or SAP Data Services.

  You can pre-calculate columns during the data provisioning phase.

Calculated Columns in Analytic Views

Calculating measures before aggregation in Analytic Views works in a way that is different from Calculation Views.

Indeed, as you have learned in the lesson, *Understanding the Deprecated Graphical Views*, the scenario of Analytic Views is made up of only three nodes: *Data Foundation*, *Star Join*, and *Semantics*, and calculated measures are defined in the Star Join node, not in the Data Foundation.

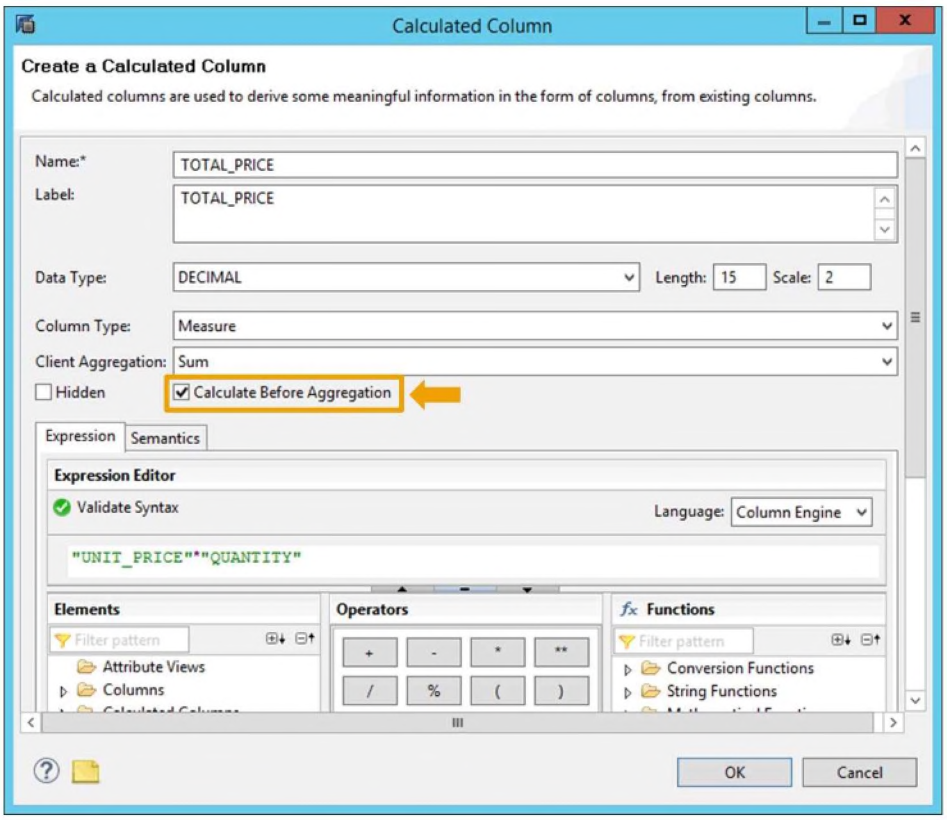


Figure 122: Calculate Before Aggregation Option in Analytic Views

As there is only one place to define calculated measures, Analytic Views provide an additional option in the Calculated Column wizard.

Expression Language and Validation

From SAP HANA SPS10 onwards, you can specify which of the two following languages the expression of a calculated column uses.

Calculated Columns: Explicit Language Definition

- SQL

  The expression only uses plain SQL.

- Column Engine

  The expression uses any valid SQLScript expression.

Validating an Expression Against a Specific Language: Key Benefits

- It helps validate the syntax of calculated columns according to the specified language.

  For example, an expression with an IF condition (supported only with SQLScript) will not validate if it is defined as SQL.

- In turn, it helps the modeler to optimize the calculation execution.

  Indeed, plain SQL expressions enable a better optimization process, compared with SQLScript. So, by validating the expression against SQL, you can make sure it will be fully optimized.

> Note:
> SAP HANA SPS11 has introduced the same distinction for filters, restricted columns, and default values expressions for variables and input parameters.

Client Side Aggregation

If you want to create a calculated measure and enable client side aggregation for the calculated measure, select the *Enable client side aggregation* checkbox.

This allows you to propose the aggregation that client needs to perform on calculated measures.

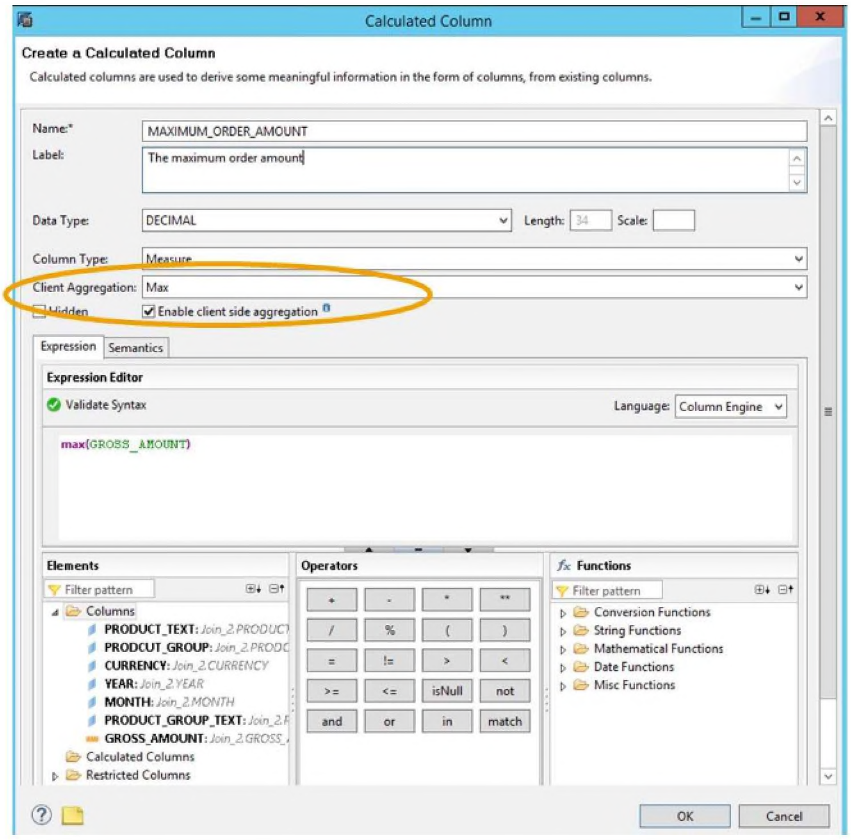


Figure 123: Client Side Aggregation

Consider an example where you have created the measure MAXIMUM_GROSS_AMOUNT giving you the maximum value of the gross amount. By selecting the *Enable client side aggregation* checkbox, you propose to the reporting client to also apply a maximum aggregation on the client side as defined in the *Client Aggregation* drop-down list.

Create Restricted and Calculated Measures

Business Example

You are a consultant at a customer that sells electronic products. The computer equipment sales are above target, but due to a very competitive market, the input device sales are below the expectations of the management. You have been asked to build individual measures to display total sales for their input device and computer sales.

The management wants to separately analyze the percentage sales share of either input device and computer sales in respect of the total company sales.

This exercise takes you through the process of creating Restricted and Calculated Measures to fulfill the reporting requirements.

> Note:
> In this exercise, when values include ##, replace the characters with the number your instructor assigned to you.

Task 1: Create a New Calculation View

1. Close all open views.
2. Create a new calculation view using the data in the following table:

Field	Value
Name:	CV_PRODUCT_SALES_ANALYSIS_##
Label:	Product sales analysis
Package:	STUDENT##
View Type:	Calculation View
Subtype:	Standard
Type:	Graphical
Data Category:	CUBE

3. In the *Calculation View* scenario, under the *Aggregation* node, add a Join node.
4. Add the following tables to the Join node:
   - TRAINING.SALES_DATA
   - TRAINING.PRODUCT
5. In your *Join* node, create a join between the tables:

Field	Value
Left Table	TRAINING.SALES_DATA
Right Table	TRAINING.PRODUCT
Join Columns	PRODUCT_ID = PRODUCT_ID
Join Type	Left Outer
Cardinality	n..1

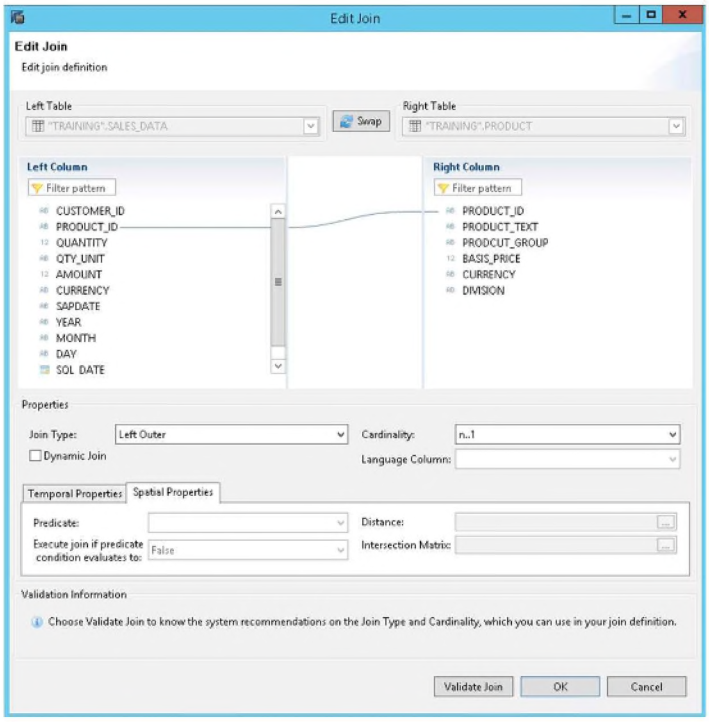


Figure 124: Edit Join

6. In the Join node, add the following columns to output:

   - SALES_DATA.AMOUNT
   - SALES_DATA.CURRENCY
   - PRODUCT.PRODUCT_TEXT

7. In the *Output* of the *Join* node, change the name property of the *AMOUNT* column to **TOTAL_SALES**.

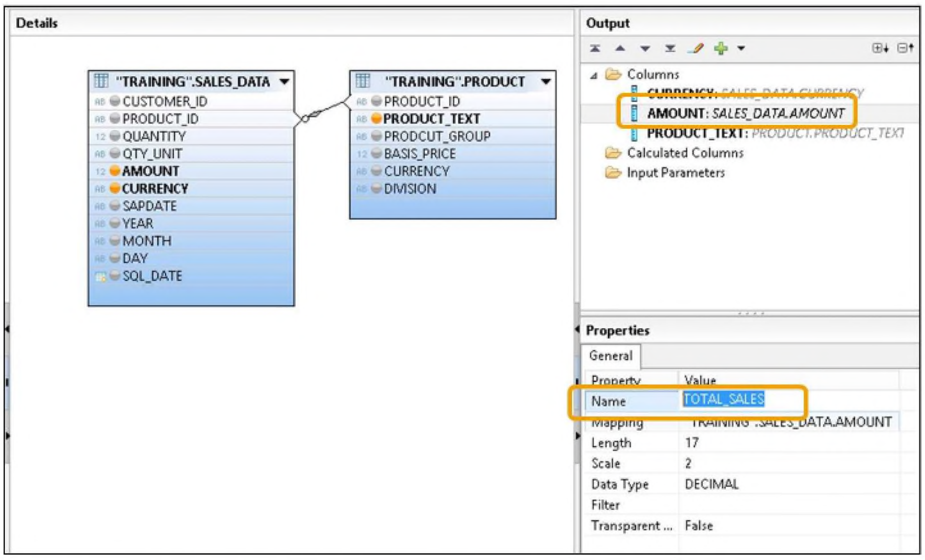


Figure 125: Change the Name Property

8. In the *Scenario*, connect the Join node to the *Aggregation* node.
9. In the *Aggregation* node, add the following columns:

Column Name	Action
CURRENCY	Add To Output
TOTAL_SALES	Add As Aggregated Column
PRODUCT_TEXT	Add To Output

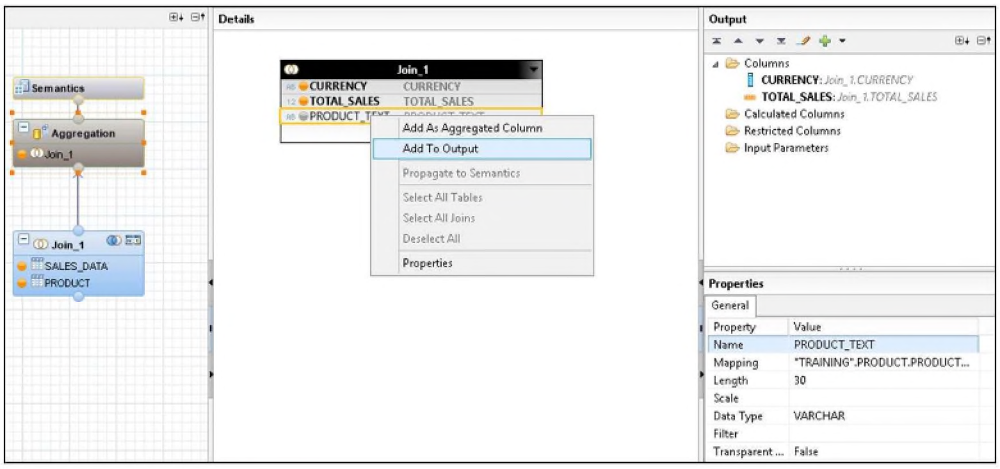


Figure 126: Add To Output

10. In the *Semantics* node, Auto-Assign columns as measures and attributes.

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

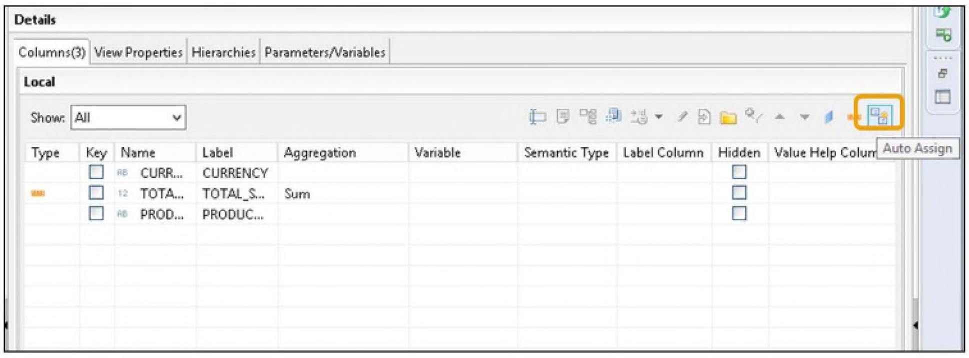


Figure 127: Auto-Assign Column Types

11. Save and Activate the calculation view.

Task 2: Create Restricted Columns by Product Type

1. In the *Aggregation* node, add a restricted column using the data in the following table:

Field	Value
Name:	INPUT_DEVICE_SALES
Label:	Total Sales for Input Devices
Column:	TOTAL_SALES

Table 7: Restrictions

Column	Operator	Value
PRODUCT_TEXT	EQUAL	Keyboard
PRODUCT_TEXT	EQUAL	Mouse

Note:
To assign value text for each of the restrictions, you can use the *Value Help Dialog* by clicking the "..." button in the value column.

2. Add another restricted column named **COMPUTER_SALES** using the data in the following table:

Field	Value
Name:	COMPUTER_SALES
Label:	Total Sales for Workstations and Laptops
Column:	TOTAL_SALES

3. Define the restrictions using an SQL expression.

   The restriction should be based on the PRODUCT_TEXT column and include:

- **`Laptop`**
- Any product text containing the word **`station`**

> Note:
> To enter a SQL expression, choose *Expression* in the *Restrictions* of the *Restricted* Column and select **`SQL`** from the *Language* drop-down menu of the Expression Editor.

The SQL expression could be written as:

```
"PRODUCT_TEXT" = 'Laptop' OR "PRODUCT_TEXT" like '%station%'
```

4. Save and activate the calculation view.
5. Display the results as Raw Data.
6. Close the *Data Preview* screen.

Task 3: Create Calculated Columns for the Sales Percentage by Product Type

1. Add a calculated column for the input device sales share to the CV_PRODUCT_SALES_ANALYSIS_## Calculation View using the data in the following table:

Field	Value
Name	INPUT_DEVICE_SALES_SHARE
Label	Percentage Share of Input Devices out of the Total Sales
Data Type	• DECIMAL • Length: 4 • Scale: 2
Column Type	Measure

2. Add the following expression to the calculated column and validate the syntax:

```
"INPUT_DEVICE_SALES" / "TOTAL_SALES"
```

> Hint:
> Instead of writing the expressions, you can double-click the column names from the *Elements* pane at the bottom of the screen, or drag them to the *Expression Editor*.
>
> You can also use CTRL + SPACE to get a restricted list of elements (columns, input parameters, and so on) or functions based on your entry or to have your entry automatically completed.

3. Add a calculated column for the computer sales share using the data in the following table:

Field	Value
Name	COMPUTER_SALES_SHARE
Label	Percentage Share of Computers out of the Total Sales
Data Type	• DECIMAL • Length: 4 • Scale: 2
Column Type	Measure

> Note:
> You can either create the new measure from scratch, or copy the existing calculated measure by:
>
> - In the *Calculated Columns* folder of the *Output* pane, right-click the *INPUT_DEVICE_SALES_SHARE* measure, select *Copy* from the context menu.
> - Right-click *Calculated Columns* folder, select *Paste* from the context menu.

4. Add the following expression to the calculated column and validate the syntax:

```
"COMPUTER_SALES" / "TOTAL_SALES"
```

5. Save and Activate the view.
6. Display a Data Preview of the results in the *Analysis* tab, showing the data in *Grid* mode using:
   - Labels axis: CURRENCY
   - Values axis: (all measures)

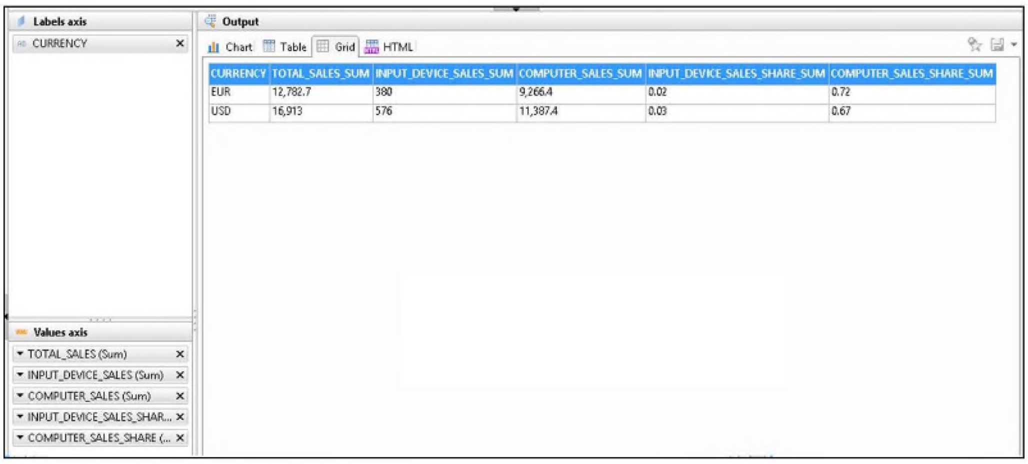


Figure 128: Grid Mode Data Preview

Unit 2
Solution 10

Create Restricted and Calculated Measures

Business Example

You are a consultant at a customer that sells electronic products. The computer equipment sales are above target, but due to a very competitive market, the input device sales are below the expectations of the management. You have been asked to build individual measures to display total sales for their input device and computer sales.

The management wants to separately analyze the percentage sales share of either input device and computer sales in respect of the total company sales.

This exercise takes you through the process of creating Restricted and Calculated Measures to fulfill the reporting requirements.

> Note:
> In this exercise, when values include ##, replace the characters with the number your instructor assigned to you.

Task 1: Create a New Calculation View

1. Close all open views.

2. Create a new calculation view using the data in the following table:

Field	Value
Name:	CV_PRODUCT_SALES_ANALYSIS_##
Label:	Product sales analysis
Package:	STUDENT##
View Type:	Calculation View
Subtype:	Standard
Type:	Graphical
Data Category:	CUBE

   a) Select *New* → *Calculation View* from the context menu of your package in the *Content* node.

   b) Enter the view properties as shown in the table.

   c) Choose *Finish*.

3. In the *Calculation View* scenario, under the *Aggregation* node, add a Join node.

   a) In the *Scenario* Palette, choose the *Join* node.

b) In the *Scenario*, place a new Join node under the *Aggregation* node.

4. Add the following tables to the Join node:

- TRAINING.SALES_DATA
- TRAINING.PRODUCT

a) Drag the above tables to the *Join* node from the TRAINING schema of the *Catalog* node.

5. In your *Join* node, create a join between the tables:

Field	Value
Left Table	TRAINING.SALES_DATA
Right Table	TRAINING.PRODUCT
Join Columns	PRODUCT_ID = PRODUCT_ID
Join Type	Left Outer
Cardinality	n..1

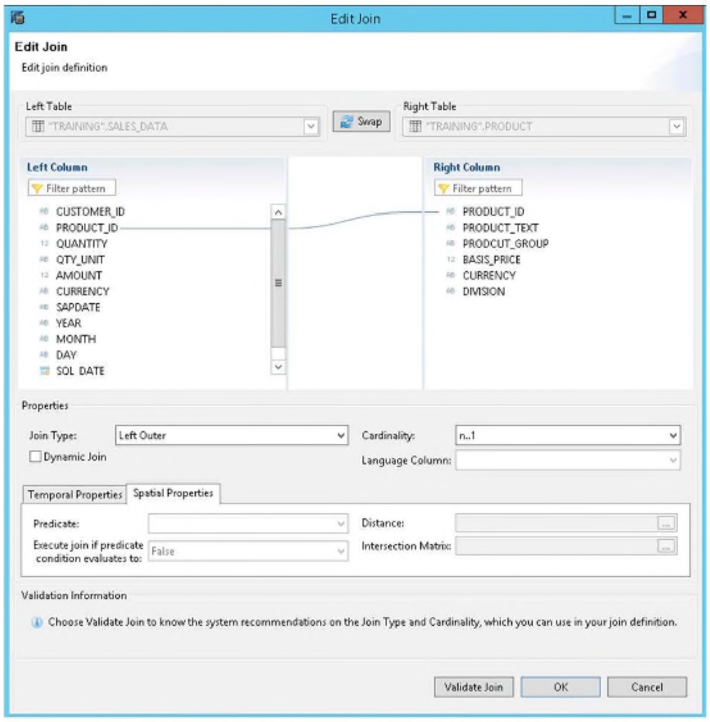


Figure 124: Edit Join

a) Drag a connecting line between the *TRAINING.SALES_DATA.PRODUCT_ID* and *TRAINING.PRODUCT.PRODUCT_ID* columns.

b) Double-click the join connector and define the properties as shown in the table.

6. In the Join node, add the following columns to output:

- SALES_DATA.AMOUNT
- SALES_DATA.CURRENCY

- PRODUCT.PRODUCT_TEXT

a) In the *Join* node, right-click the above columns and select *Add To Output* from the context menu.

7. In the *Output* of the *Join* node, change the name property of the *AMOUNT* column to **TOTAL_SALES**.

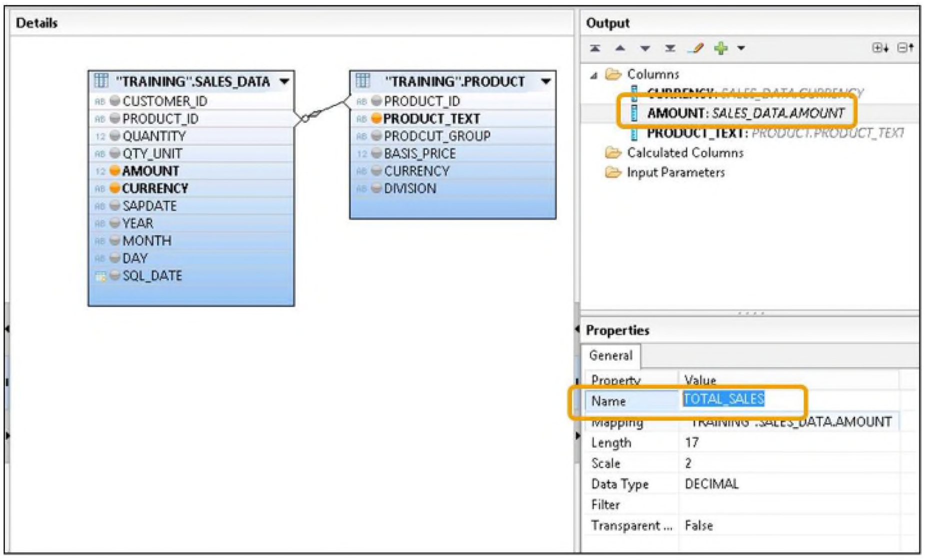


Figure 125: Change the Name Property

a) In the *Output* of the *Join* node, select the *AMOUNT* column.

b) Change the *Name* property to **TOTAL_SALES**.

8. In the *Scenario*, connect the Join node to the *Aggregation* node.

a) In the Scenario of the calculation view, drag a connecting line from the *Join* node to the *Aggregation* node.

9. In the *Aggregation* node, add the following columns:

Column Name	Action
CURRENCY	Add To Output
TOTAL_SALES	Add As Aggregated Column
PRODUCT_TEXT	Add To Output

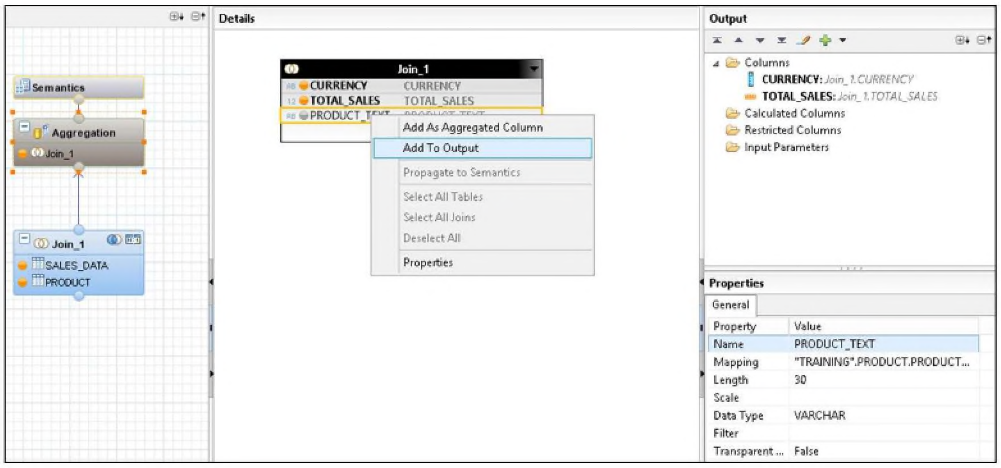


Figure 126: Add To Output

a) In the *Aggregation* node, right-click each column and select *Add To Outut/Add As Aggregated Column* as shown in the table.

10. In the *Semantics* node, Auto-Assign columns as measures and attributes.

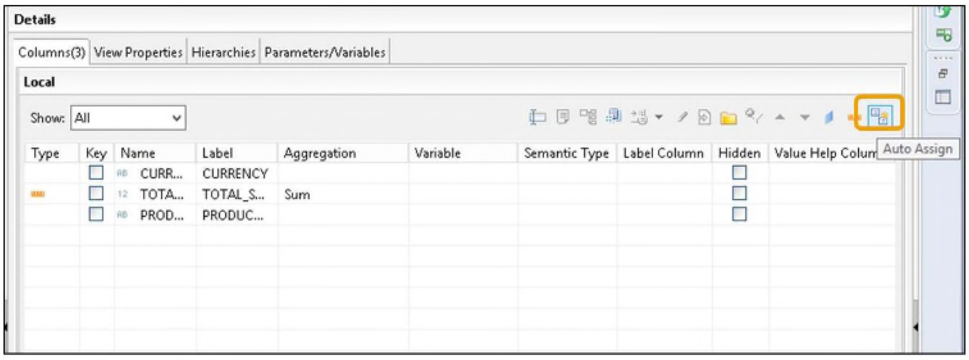


Figure 127: Auto-Assign Column Types

a) In the *Columns* tab of the *Semantics* node, choose *Auto-Assign*.

11. Save and Activate the calculation view.

Task 2: Create Restricted Columns by Product Type

1. In the *Aggregation* node, add a restricted column using the data in the following table:

Field	Value
Name:	INPUT_DEVICE_SALES
Label:	Total Sales for Input Devices
Column:	TOTAL_SALES

Table 7: Restrictions

Column	Operator	Value
PRODUCT_TEXT	EQUAL	Keyboard
PRODUCT_TEXT	EQUAL	Mouse

> Note:
> To assign value text for each of the restrictions, you can use the *Value Help Dialog* by clicking the "..." button in the value column.

a) In the *Output* of the *Aggregation* node, right click *Restricted Columns* and select *New...* from the context menu.

b) Enter the data as shown in the table.

c) Choose *OK*.

2. Add another restricted column named **COMPUTER_SALES** using the data in the following table:

Field	Value
Name:	COMPUTER_SALES
Label:	Total Sales for Workstations and Laptops
Column:	TOTAL_SALES

a) In the *Output* of the *Aggregation* node, right-click *Restricted Columns* and select *New...* from the context menu.

b) Enter the data from the above table.

3. Define the restrictions using an SQL expression.

The restriction should be based on the PRODUCT_TEXT column and include:

- **Laptop**
- Any product text containing the word **station**

> Note:
> To enter a SQL expression, choose *Expression* in the *Restrictions* of the *Restricted* Column and select **SQL** from the *Language* drop-down menu of the Expression Editor.

The SQL expression could be written as:

```
"PRODUCT_TEXT" = 'Laptop' OR "PRODUCT_TEXT" like '%station%'
```

a) In *Restrictions*, select *Expression*.

b) In the *Language* drop-down menu, select *SQL*.

c) Enter the following as the expression:

```
"PRODUCT_TEXT" = 'Laptop' OR "PRODUCT_TEXT" like '%station%'
```

d) Choose *OK*.

4. Save and activate the calculation view.

a) Choose *Save and Activate*.

5. Display the results as Raw Data.
   a) Choose *Data Preview*.
   b) Select the *Raw Data* tab.
6. Close the *Data Preview* screen.

Task 3: Create Calculated Columns for the Sales Percentage by Product Type

1. Add a calculated column for the input device sales share to the CV_PRODUCT_SALES_ANALYSIS_## Calculation View using the data in the following table:

Field	Value
Name	INPUT_DEVICE_SALES_SHARE
Label	Percentage Share of Input Devices out of the Total Sales
Data Type	• DECIMAL • Length: 4 • Scale: 2
Column Type	Measure

   a) In the *Aggregation* node, in the *Output* view, right-click the *Calculated Columns* folder and choose *New Calculated Column*.
   b) In the *Calculated Column* dialog box, enter the data as shown in the table.
2. Add the following expression to the calculated column and validate the syntax:

```
"INPUT_DEVICE_SALES" / "TOTAL_SALES"
```

> Hint:
> Instead of writing the expressions, you can double-click the column names from the *Elements* pane at the bottom of the screen, or drag them to the *Expression Editor*.
>
> You can also use CTRL + SPACE to get a restricted list of elements (columns, input parameters, and so on) or functions based on your entry or to have your entry automatically completed.

   a) In the *Calculated Columns* dialog box, on the *Expression* tab, in the *Expression Editor*, enter the listed expression.
   b) Choose *Validate Syntax*.
   c) Choose *OK*.
3. Add a calculated column for the computer sales share using the data in the following table:

Field	Value
Name	COMPUTER_SALES_SHARE

Field	Value
Label	Percentage Share of Computers out of the Total Sales
Data Type	• DECIMAL • Length: 4 • Scale: 2
Column Type	Measure

> Note:
> You can either create the new measure from scratch, or copy the existing calculated measure by:
>
> - In the *Calculated Columns* folder of the *Output* pane, right-click the *INPUT_DEVICE_SALES_SHARE* measure, select *Copy* from the context menu.
> - Right-click *Calculated Columns* folder, select *Paste* from the context menu.

a) In the *Aggregation* node, in the *Output* view, right-click the *Calculated Columns* folder and choose *New Calculated Column*.

b) In the *Calculated Column* dialog box, enter the data from the table.

4. Add the following expression to the calculated column and validate the syntax:

```
"COMPUTER_SALES" / "TOTAL_SALES"
```

a) In the *Calculated Columns* dialog box, on the *Expression* tab, in the *Expression Editor*, enter the listed expression.

b) Choose *Validate Syntax*.

c) Choose *OK*.

5. Save and Activate the view.

a) Choose *Save and Activate*.

6. Display a Data Preview of the results in the *Analysis* tab, showing the data in *Grid* mode using:

- Labels axis: CURRENCY
- Values axis: (all measures)

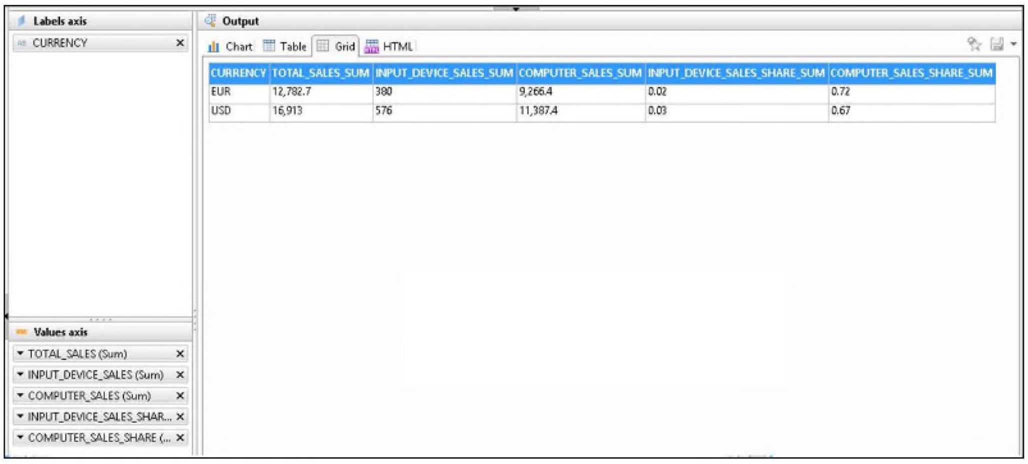


Figure 128: Grid Mode Data Preview

a) Choose *Data Preview*.

b) Drag the CURRENCY attribute to *Labels Axis*.

c) Drag all the measures to *Values Axis*.

LESSON SUMMARY

You should now be able to:

- Create restricted and calculated columns
- Create Restricted Columns
- Create calculated columns

Unit 2
Lesson 3

Filtering Data

LESSON OVERVIEW

Most of the time, users do not need the whole data perimeter stored in the source system. In these instances, filters can be used to display only relevant data. Moreover, by decreasing data volume, performance can be improved.

Business Example

You notice that a large amount of data stored in base tables are never used for reporting, because users decide to apply WHERE clause filters in reporting tools.

In order to speed up query execution you decide to introduce filters into information models.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Filter data
- Create client dependent views
- Restrict data when modeling using domain fix values

Using Filter Operations

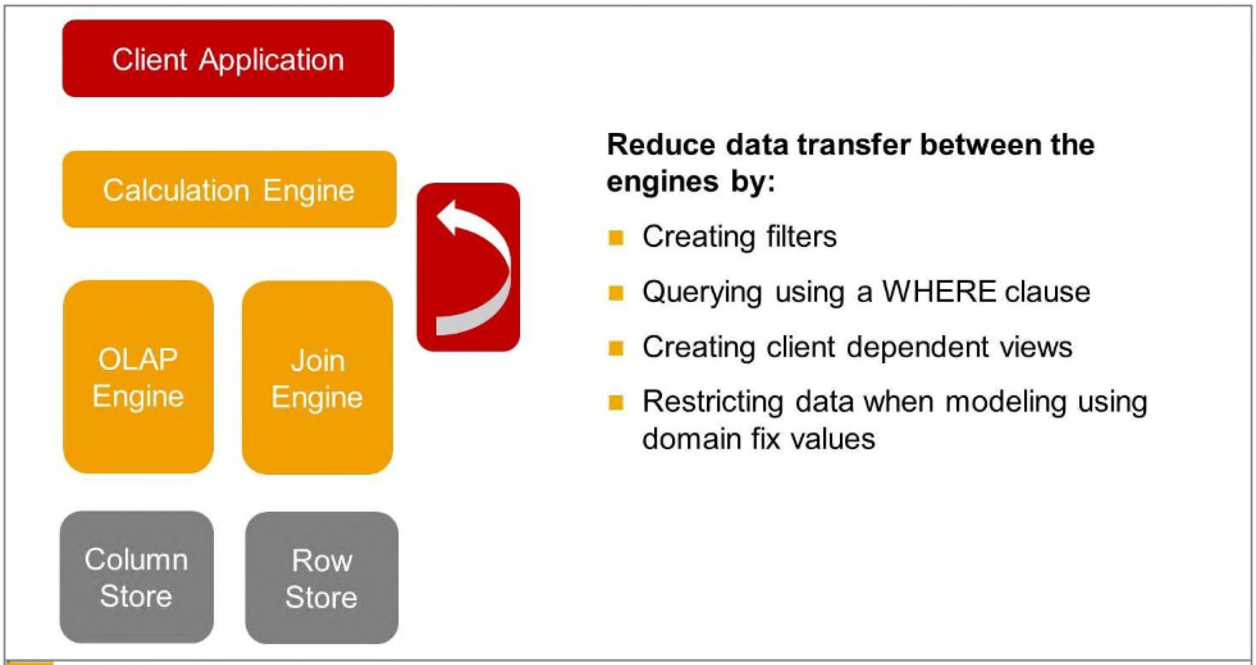


Figure 129: Filter Operations

A primary goal during modeling is to minimize data transfers between engines. This statement holds true both internally, that is, in the database between engines, but also between SAP HANA and the end user client application. For example, an end user will never

need to display million rows of data. Such a large amount of information just cannot be consumed in a meaningful way.

Whenever possible, data should be aggregated and filtered to a manageable size before it leaves the data layer.

When deciding which records should be reported upon, a best practice is to think at a set level not a record level.

A set of data can be aggregated by a region, a date, or some other group to minimize the amount of data passed between views.

Comparing Filters and WHERE Clauses

Both WHERE clauses and filters are used to reduce the result set of data. However, from a semantic perspective, these two operations are not the same.

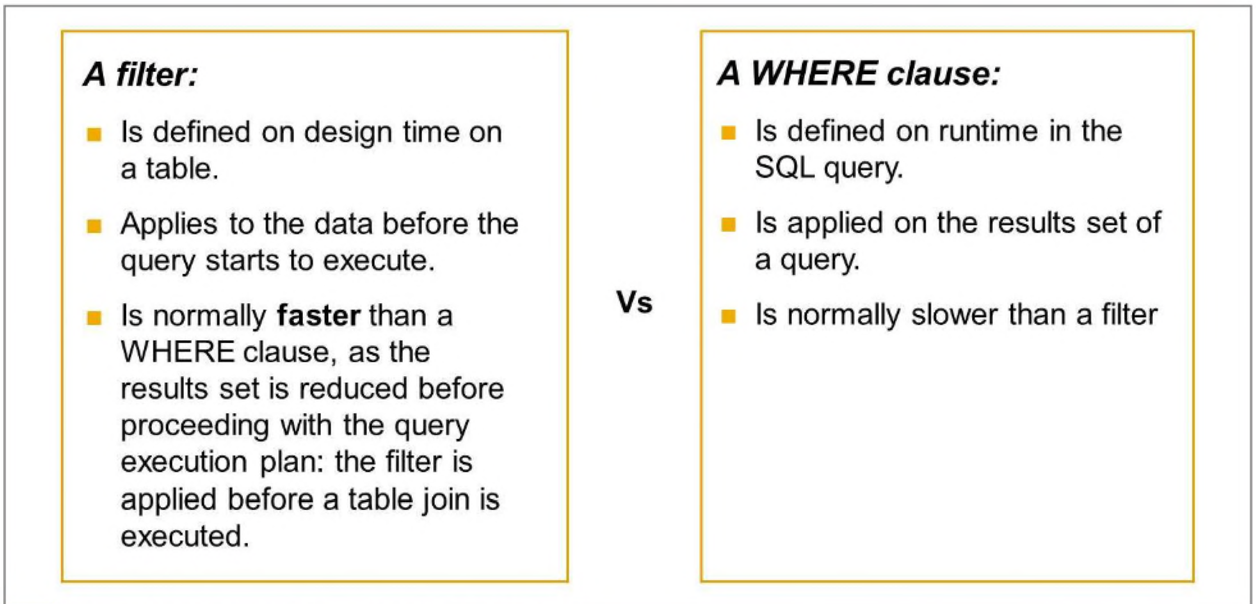
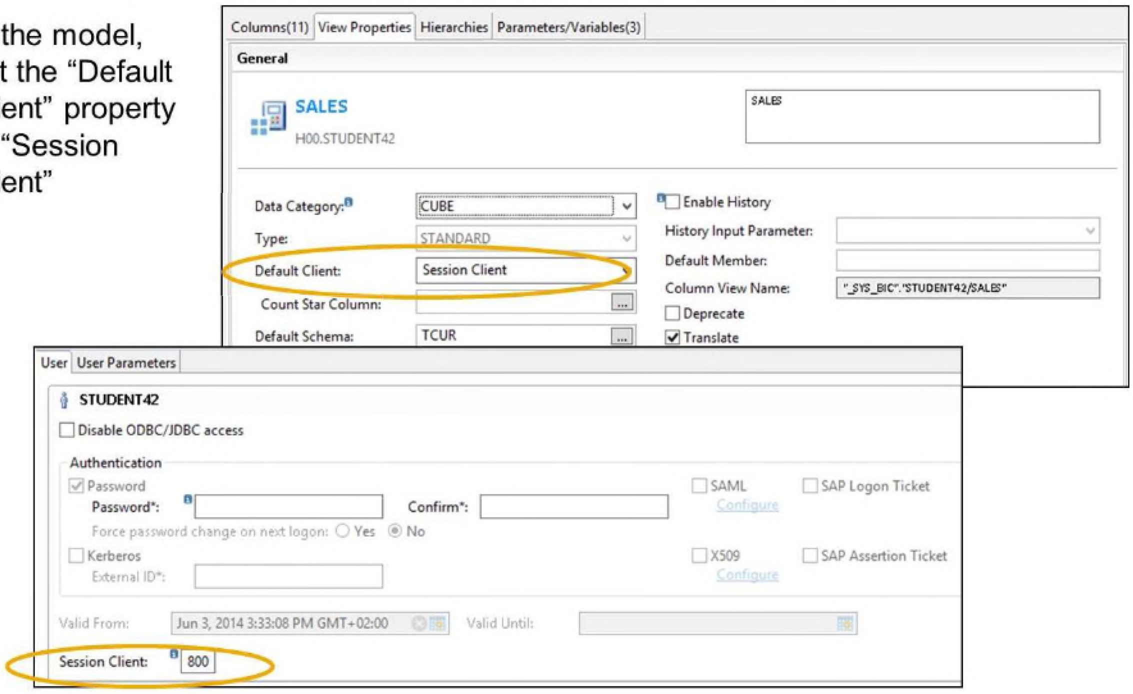


Figure 130: Comparing Filters and WHERE Clauses

Client Dependent Views

- In the model, set the “Default Client” property to “Session Client”



- In Security, assign a “Session Client”.

➡ When the user runs a query based on this model, the user’s Session Client (in this example: 800) is used to filter the source data.

Figure 131: Create Client Dependent Views

Almost all tables in the SAP Business Suite data model are client-dependent.

Only a small number of tables do not have the client as the first primary key field.

Information model data is automatically filtered as follows:

- To one client by setting the *Default Client* property in the model to a fixed client value. It is also automatically filtered
- To one client at runtime for information models where the *Default Client* property in the model is set to *Session Client*, so that the client value assigned to the user in their user id is used as the filter.
- To one client at runtime for information models where a filter has been defined against the column *MANDT* or *CLIENT* using the system variable $$client$$. This variable picks up the client value stored in the user’s ID.

Handling Users with no Default Client

When an information view has its *Default Client* property set to *Session Client*, but the connected user does not have a CLIENT number assigned (that is, the *Session Client* property of the user is blank), SAP HANA behaves differently depending on the type of view.

View Behavior when the User has no Assigned CLIENT

- In Calculation Views:

  The view does not retrieve any data

- In Analytic Views:

  The view retrieves data for all CLIENT values. That is, behaves as if its *Default Client* property were set to *Cross Client*.

Session Dependent Functions

To select dynamically table data in Calculation Views, you can use the following functions. These functions return client or locale information based for the connected user.

Session Dependent Functions

- SESSION_CONTEXT('CLIENT')

  Returns the client's value based on the current user profile.

- SESSION_CONTEXT('LOCALE')

  Returns the session's language in POSIX format (set by 'locale' parameter of JDBC/ODBC/ODBO connection, for example, en_en, de_de, de_at).

- SESSION_CONTEXT('LOCALE_SAP')

  Returns the session's language following the SAP internal format (like the SPRAS column in the text tables of Master Data).

Example:

```
SELECT field1, field2 FROM my_table WHERE mandt =
SESSION_CONTEXT('CLIENT') AND spras = SESSION_CONTEXT('LOCALE_SAP')
```

Domain Fix Values

Domain and Domain Fix values are data concepts within SAP ECC that need to be taken into account when modeling using data from SAP ECC.

Definition of a Domain

A domain defines a value range. A domain is assigned to a data element. All table fields or structure components that use this data element then have the value range defined by the domain. The relationship between the field or component and the domain is defined by the data element of the field or component.

Fields or components that refer to the same domain (with the assigned data elements) are also changed when the domain is changed. This ensures that the value ranges of these fields or components are consistent. Fields or components that are technically the same can be combined with a reference to the same domain.

The value range of a domain can be restricted by defining fixed values. If all the fields or components that refer to the domain should be checked against a certain table, this table can be defined as the value table of the domain.

Modeling Using Domain Fix Values

Domain fix values can be used for several fields, which are simple value lists with language-dependent texts. Actual values are stored in database tables (DD07L, DD07T).

When using domain fix values, it can be useful to restrict data. Filters can be applied on the language-dependent fields using a text join.

During the initialization of replication, two tables are created for each used domain. Only active domain values are transferred (as identified by field AS4LOCAL = 'A' in DD07T).

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

The naming convention for these tables is DOM_<Domain Name> for the Domain values and DOM_<Domain Name>T for the language-dependent texts. Examples include DOM_MEPRF and DOM_MEPRFT, DOM_BFART and DOM_BFARTT.

Domain fix value tables are located in the same schema as the replicated tables.

For example, you can build views on these tables so that they can be used in calculation views. Both related tables must be included together with a text join on language field DDLANGUAGE. Include field DOMVALUE_L as key field in the output structure and add the field DDTEXT.

- In order to restrict data, you define a text join and a cardinality 1..1.
- Language Column is defined as DDLANGUAGE column.

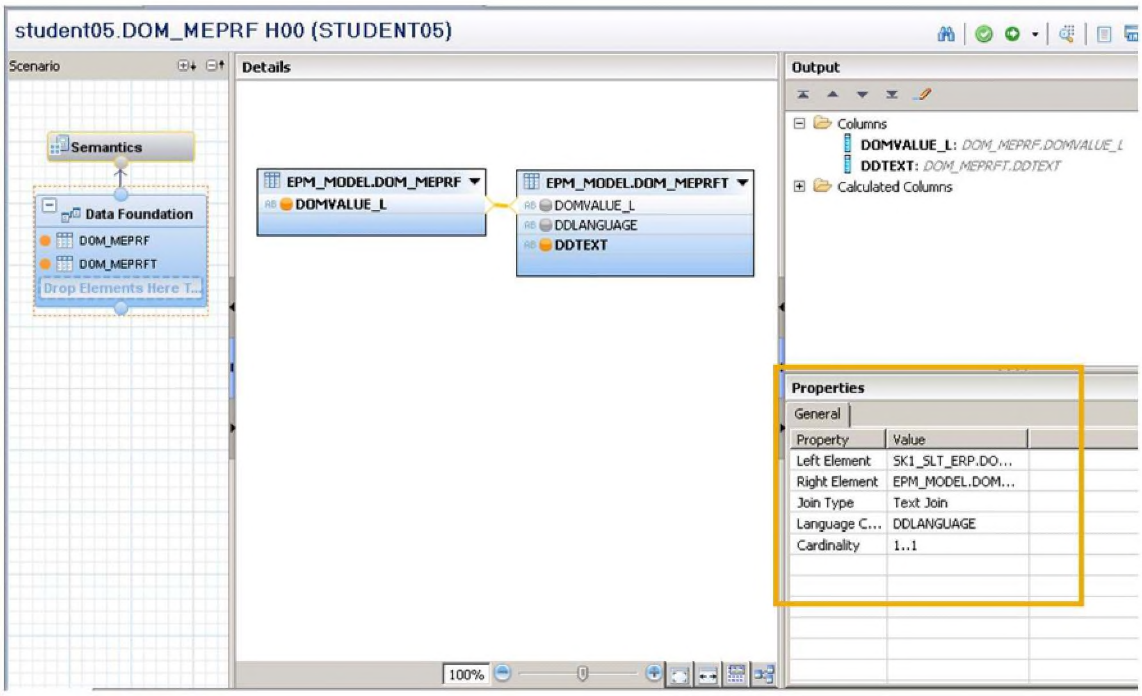


Figure 132: Restricting Domain Fix Values Using Text Join

Unit 2
Exercise 11

Define and Use Filters

Business Example

You want to use the CLIENT dimension (also known as MANDT) to retrieve consistent data. You also want to apply a filter to an attribute to reduce the set of data retrieved by the view.

Note:
For participants who might not be familiar with the classical database structure of solutions such as SAP Business Suite, please note that the concept of CLIENT does not relate to any customer, but is generally intended to distinguish different sets of data (for example, *Development*, *Quality Assurance*, *Training*) within the same non-productive SAP system (same schema in the same database).

On production systems, it is highly recommended to have only one CLIENT.

Note:
In this exercise, when values include ##, replace the characters with the number your instructor assigned to you.

Task 1: Create a New Calculation View to Show Data Across Different Clients

1. Close all open views.
2. Create a new calculation view using the data in the following table:

Field	Value
Name:	CV_CITIES_TRAIN_##
Label:	Cities
Package:	STUDENT##
View Type:	Calculation View
Subtype:	Standard
Type:	Graphical
Data Category:	DIMENSION

3. In the calculation view scenario, under the *Projection* node, add another Projection node.
4. Add the following table to the new Projection node:
   - TRAINING.CITIES
5. Add all columns to the output of the *Projection* node.

6. Connect the new *Projection* node to the uppermost *Projection* node.
7. Add all columns to the output of the uppermost *Projection* node.
8. In the *Semantics* node, set the *Default Client* property in the *View Properties* tab to *Cross Client*.
9. Save and activate the calculation view.

> Note:
> An intermediate solution object is available at this stage at *STUDENT00 → CV_CITIES_TRAIN00_STAGE1*.

10. Preview the calculation view data in the *Raw Data* tab.

> Note:
> The column MANDT contains the CLIENT number.

11. In the Data Preview, on the *Distinct Values* tab, select *CITY* from the attributes. How many distinct values do you see?

> Caution:
> Modeling without paying attention to CLIENT/MANDT can cause undesired or incorrect results

Task 2: Modify the Default Client Setting to Only Display Data from the Appropriate Client

1. Modify your calculation view by changing the *Default Client* property to *Session Client*.
2. Save and activate the calculation view.

> Note:
> An intermediate solution object is available at this stage at *STUDENT00 → CV_CITIES_TRAIN00_STAGE2*.

3. Preview the calculation view data in the *Raw Data* tab.

> Note:
> In the users table of the SAP HANA database, your user STUDENT## has been assigned a default client value of 800, which means that whenever you execute a view that has the Default Client property set to Session Client, this parameter is passed to the view to filter the data and extract only the data for which the CLIENT column equals 800.

4. In the Data Preview, on the *Distinct Values* tab, select *CITY* from the attributes. How many distinct values do you see?

Task 3: Apply a Filter to your Calculation View to Only Show Capital Cities

1. In the lowermost *Projection* node, using the *CAPITAL* column, apply a filter using the data from the following table:

Field	Value
Operator	Equal
Value	Y

> Note:
> Following this change, your filter selects only one value, and you may choose to remove the *CAPITAL* column from the output.

2. Save and activate the calculation view.

3. Preview the calculation view data in the Raw Data tab.

4. In the Data Preview, on the *Distinct Values* tab, select *CITY* from the attributes. How many distinct values do you see?

Unit 2
Solution 11

Define and Use Filters

Business Example

You want to use the CLIENT dimension (also known as MANDT) to retrieve consistent data. You also want to apply a filter to an attribute to reduce the set of data retrieved by the view.

> Note:
> For participants who might not be familiar with the classical database structure of solutions such as SAP Business Suite, please note that the concept of CLIENT does not relate to any customer, but is generally intended to distinguish different sets of data (for example, *Development*, *Quality Assurance*, *Training*) within the same non-productive SAP system (same schema in the same database).
>
> On production systems, it is highly recommended to have only one CLIENT.

> Note:
> In this exercise, when values include ##, replace the characters with the number your instructor assigned to you.

Task 1: Create a New Calculation View to Show Data Across Different Clients

1. Close all open views.
2. Create a new calculation view using the data in the following table:

Field	Value
Name:	CV_CITIES_TRAIN_##
Label:	Cities
Package:	STUDENT##
View Type:	Calculation View
Subtype:	Standard
Type:	Graphical
Data Category:	DIMENSION

   a) Select *New* → *Calculation View* from the context menu of your package in the *Content* node.

   b) Enter the view properties as shown in the table.

   c) Choose *Finish*.

3. In the calculation view scenario, under the *Projection* node, add another Projection node.
   a) In the *Scenario* Palette, choose the *Projection* node.

   b) In the *Scenario*, place a new *Projection* node under the existing Projection node.

4. Add the following table to the new Projection node:

   - TRAINING.CITIES

   a) Drag the table to the *Projection* node from the TRAINING schema of the *Catalog* node.

5. Add all columns to the output of the *Projection* node.
   a) In the *Details* pane of the *Projection* node, right-click the table *"TRAINING".CITIES* and choose *Add All To Output* from the context menu.

6. Connect the new *Projection* node to the uppermost *Projection* node.
   a) Drag a connecting line from the new *Projection* node to the uppermost *Projection* node.

7. Add all columns to the output of the uppermost *Projection* node.
   a) In the *Details* pane of the uppermost *Projection* node, right-click the node/table and choose *Add All To Output* from the context menu.

8. In the *Semantics* node, set the *Default Client* property in the *View Properties* tab to *Cross Client*.
   a) In the *Scenario*, select the *Semantics* node.

   b) In the *Default Client* drop-down list of the *View Properties* tab, choose *Cross Client*.

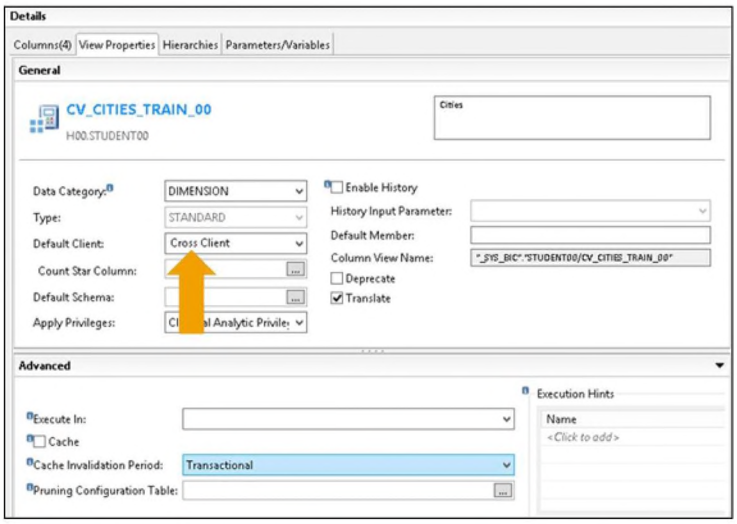


Figure 133: Setting the Default Client Property

9. Save and activate the calculation view.

> **Note:**
> An intermediate solution object is available at this stage at *STUDENT00 → CV_CITIES_TRAIN00_STAGE1*.

   a) Choose *Save and Activate*.

10. Preview the calculation view data in the *Raw Data* tab.

> Note:
> The column MANDT contains the CLIENT number.

a) Choose *Data Preview*.

b) Select the *Raw Data* tab and review the contents.

11. In the Data Preview, on the *Distinct Values* tab, select *CITY* from the attributes. How many distinct values do you see?

20

> Caution:
> Modeling without paying attention to CLIENT/MANDT can cause undesired or incorrect results

Task 2: Modify the Default Client Setting to Only Display Data from the Appropriate Client

1. Modify your calculation view by changing the *Default Client* property to *Session Client*.

   a) In the *Semantics* node, select the *View Properties* tab.

   b) In the *Default Client* drop-down list, select *Session Client*.

2. Save and activate the calculation view.

> Note:
> An intermediate solution object is available at this stage at *STUDENT00 → CV_CITIES_TRAIN00_STAGE2*.

3. Preview the calculation view data in the *Raw Data* tab.

> Note:
> In the users table of the SAP HANA database, your user STUDENT## has been assigned a default client value of 800, which means that whenever you execute a view that has the Default Client property set to Session Client, this parameter is passed to the view to filter the data and extract only the data for which the CLIENT column equals 800.

   a) Choose *Data Preview*.

   b) Select the *Raw Data* tab and review the contents.

4. In the Data Preview, on the *Distinct Values* tab, select *CITY* from the attributes. How many distinct values do you see?

   8

Task 3: Apply a Filter to your Calculation View to Only Show Capital Cities

1. In the lowermost *Projection* node, using the *CAPITAL* column, apply a filter using the data from the following table:

Field	Value
Operator	Equal
Value	Y

> Note:
> Following this change, your filter selects only one value, and you may choose to remove the *CAPITAL* column from the output.

   a) In the lowermost *Projection* node of the Scenario, right-click the *CAPITAL* column, and choose *Apply Filter* from the context menu.

   b) Enter the data from the above table to define the filter.

   c) Choose *OK*.

   d) Choose *OK*.

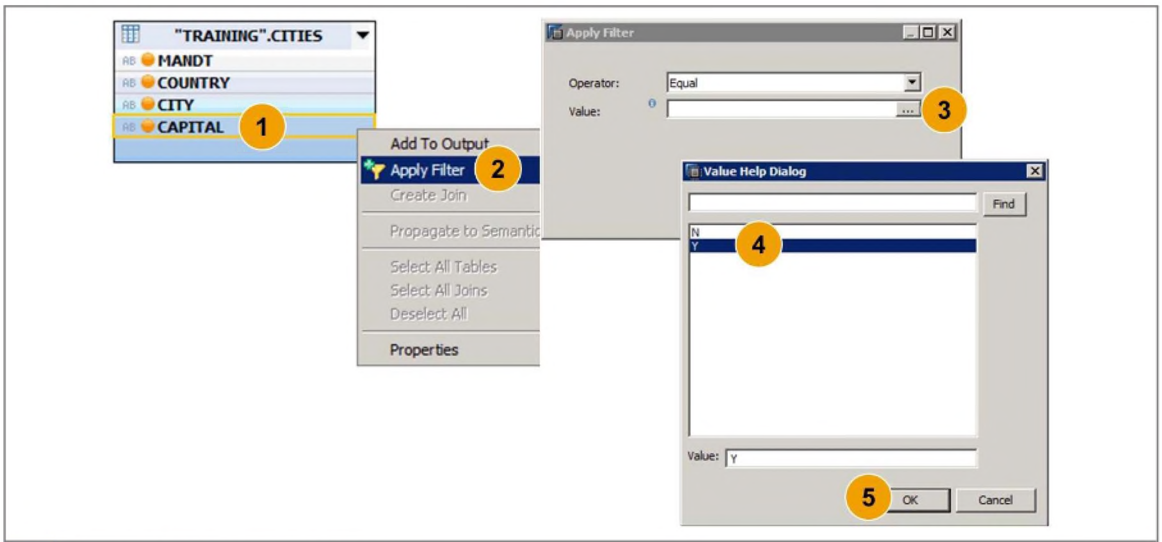


Figure 134: Applying a Filter

2. Save and activate the calculation view.

3. Preview the calculation view data in the Raw Data tab.

   a) Choose *Data Preview*.

   b) Select the *Raw Data* tab and review the contents.

   c) Select the *Raw Data* tab and review the contents.

4. In the Data Preview, on the *Distinct Values* tab, select *CITY* from the attributes. How many distinct values do you see?

   Answers will vary.

LESSON SUMMARY

You should now be able to:

- Filter data
- Create client dependent views
- Restrict data when modeling using domain fix values

Unit 2
Lesson 4

Using Variables and Input Parameters

LESSON OVERVIEW

This lesson explains how you can customize your information views by creating variables and input parameters.

These artifacts will result in dedicated dialog boxes that will appear when the view is executed, requesting the end user to pass parameters to the query to modify the result set or trigger parameter-based calculations.

These artifacts provide the flexibility to use the same information view with parameters entered at runtime, instead of hard-coding such parameters in different views during the design. It also helps restrict the set of data that is transferred between SAP HANA and the reporting tools.

You can also compute calculated columns based on a parameter entered by the user at runtime instead of hard-coding it in the information view itself.

Business Example

You want to build a view to report sales by product, but only for the country (or region) the user specifies when executing the view.

Additionally, you would like to create in your information models calculations based on user-defined parameters.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Use variables and input parameters
- Create variables and use them to filter data
- Create input parameters

Variables and Input Parameters

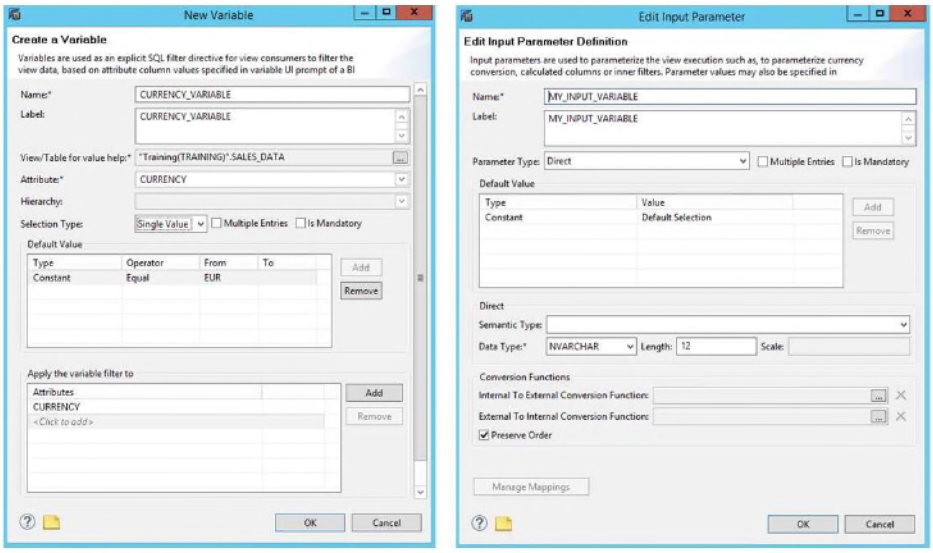
Variables and Input Parameters are features that enable the modeler to create more dynamic information models. The end user, consuming the data, will be prompted to provide attribute values or parameters, either to reduce the result set of a view, or to trigger calculations based on the provided parameters.

Difference Between Variables and Input Parameters

Variables and Input parameters in SAP HANA can be described as types of input boxes.

It is a way of asking the reporting user for input on how to provide, present, or restrict the data that will be displayed.

> Note:
> For course participants familiar with SAP Business Objects terminology, the variable or input parameter is referred to as a *Prompt* in Business Object tools.
>
> In SAP Business Warehouse, they may be referred to as *Variables*.



- **Variables**

  Bound to attributes and are used for filtering.

- **Input Parameters**

  Can contain any value a reporting user enters. These are typically used for formulas.

Figure 135: Variables Compared to Input Parameters (I)

Difference between Variables and Input Parameters (2)

Variables

- Variables are used to filter the content of an information model. They do not impact the execution workflow of the information model and are applied to a query to filter out some values.
- Variables can be used to filter attributes. As an example, a variable can be used to filter a result to a specific country or a specific customer.
- When filtering using variables, a WHERE clause is added to the SQL query.

Input parameters

- Calculations performed by a model can use input parameters as input values
- Input parameters can be defined as compulsory for the evaluation of an information model.
- In SQL, input parameter values are passed via the PLACEHOLDER reserved word.

Figure 136: Variables Compared to Input Parameters (II)

Impact of Variables and Input Parameters



- You use variables to filter data at runtime. You assign values to these variables by entering the value manually, or by selecting it from the drop-down list.
- Using variables means that you do not need to decide the restriction on the value of attributes at the design time.
- You can apply variables in information models
- If a calculation view is created using an analytic view with input parameters, those input parameters are also available in the calculation view but cannot be edited.

Figure 137: Impact of Variables and Input Parameters

Variables

The following types of Variables are supported:

Type	Description
Single Value	Use this to filter data on a single value of the column
Interval	Use this where you want the user to specify an interval between two values of the column.
Range	Use this when you want the end user to use operators such as “Greater Than” or “Less Than”.

Range Variable Example:

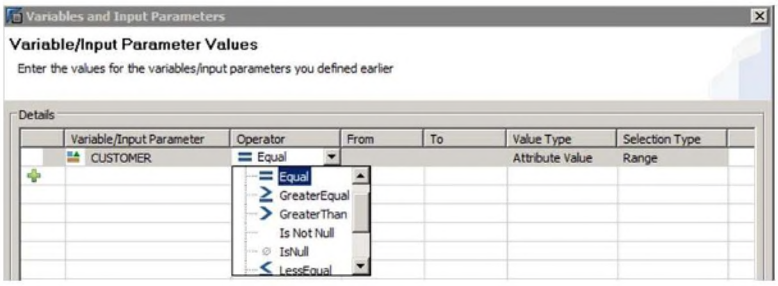
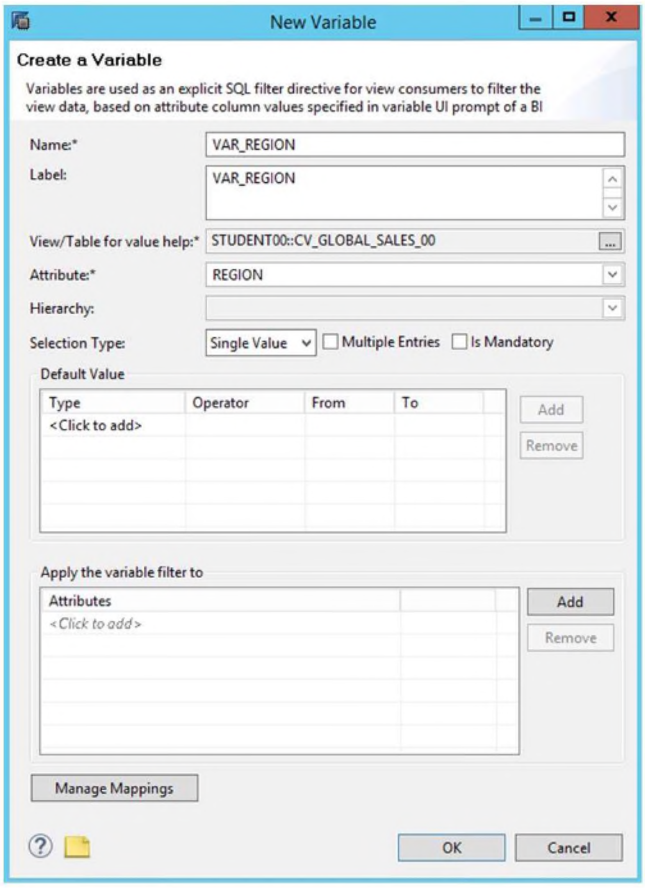


Figure 138: Variables Types

Creating Variables



A variable definition includes:

- **View/Table for Value Help and Attribute:** These settings define which view/table and which attribute from this view/table is used as a reference to provide a list of values at runtime
- **Selection Type:** Whether selections should be based on intervals, ranges or single values.
- **Multiple Entries:** Whether multiple occurrences of the selection type are allowed.
- You can also define whether specifying the variable at runtime is **Mandatory** and/or if it should have a **Default Value**.
- You define which **attribute(s)** of the current view the variable should be applied to.

Figure 139: Creating Variables (I)

The behavior of variables at runtime depends on whether an entry is required for the variable or not, see the following:

- If a variable is defined as *Mandatory*, the user needs to provide the value(s), range(s) or interval(s) at runtime.
- For non-mandatory variables, if nothing is specified at runtime, all the data for the corresponding attributes is returned by the view without filtering.

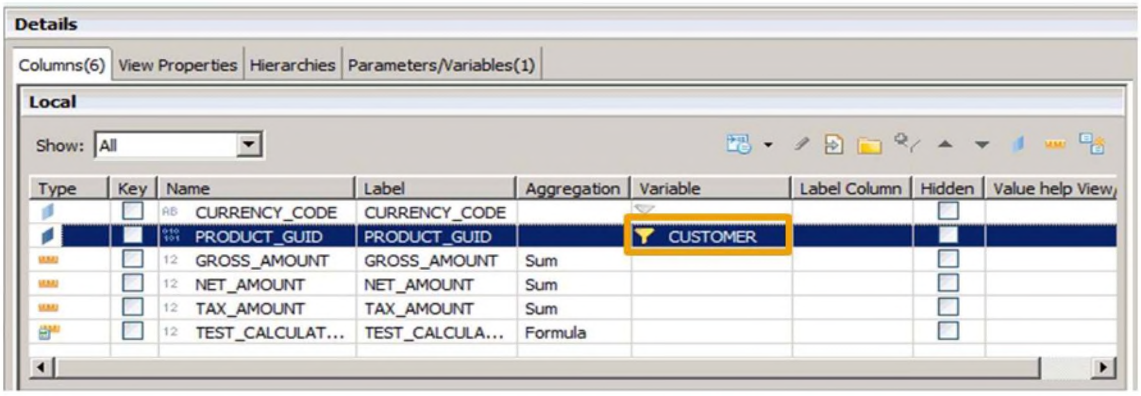
The attribute selection for *View/Table value for help* is important for performance. If you select an attribute from a table rather than the view itself, this will be faster as the provision of list of values will not need to execute the view to provide the attributes.

You can use a variable in the expression of calculated columns of the types *Attribute* or *Measure*, but only on the condition that only one value is passed to the view.

For this reason, in this kind of scenario, the only suitable way to define the variable is to assign the Selection Type *Single Value* and to prevent *Multiple Entries*.

> Caution:
> Using other ways to define the variable, either returns an error from the data provider at runtime, or the variable seems to be working but returns the wrong results.

Creating Variables (2)



- In the semantics of a view, you can see, and also define, which variable is assigned to which attribute.
- Note that one variable can be assigned to multiple attributes.

Figure 140: Creating Variables (II)

Creating Variables (3)

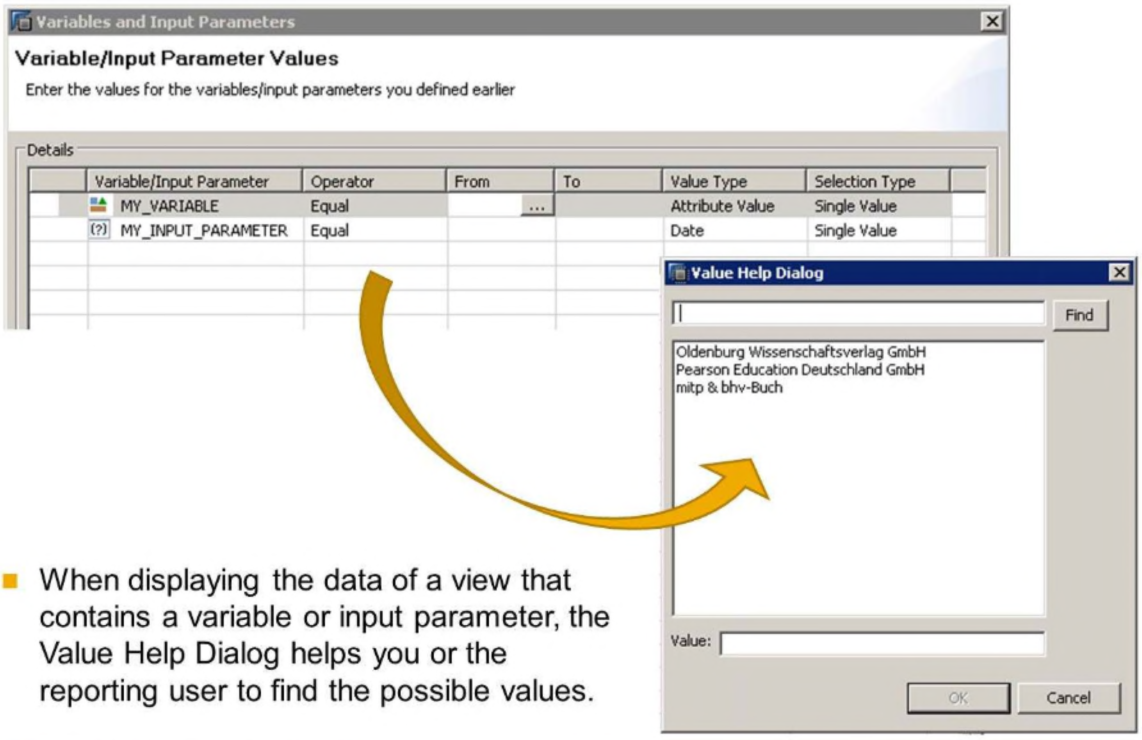


Figure 141: Creating Variables (III)

More than one value can be chosen for a variable when you select the *Multiple Entries* checkbox.

> Note:
> In the data preview, *From* and *To* are displayed in the *Variable Values* dialog even when a variable has not been defined as *range*.

Input parameters



- You might not want a variable to just restrict the data of a view.
- You might want to take input from the user and process it, returning dynamic data based on the user selection.
- Input Parameters make this possible.

Figure 142: Input Parameter Use Cases

You can use input parameters to define internal parameterization of the view. Input parameters used in the views enable you to obtain a desired functionality when the view is executed.

Input Parameter Types

The following types of Input Parameters are supported:

Type	Use Cases
Direct: Currency	For currency conversion, when you want the end user to specify a source or target currency.
Direct: Date	To retrieve data based on a date entered by the end user (or chosen in a calendar type input box).
Direct: Unit of Measure	To retrieve data based on a unit of measure choosen by the end user
Static List	To provide the end user with a predefined list of values in which he/she chooses one or several items.
Column	To provide the end user with a list of values from a column of the information model
Derived From Table	When you want the end user to have a set list of values from a table (not necessarily included in the view)
Derived From Procedure	When you want the parameter value to be passed to the information model based on the scalar output of a stored procedure
Direct (without semantic type)	When none of the above applies and/or when you want the user to enter a parameter without choosing it from a predefined list.

Figure 143: Input Parameter Types

The figure, Input Parameter Types, shows the different types of Input Parameters that can be defined.

The *Direct* parameter type can be combined with a semantic type such as *Date*, *Currency*, or *Unit of measure*.

For the *Currency* and *Unit of measure* semantic types, the list of proposed values will be created based on the corresponding reference tables in SAP HANA. This setup requires that the default schema assigned to the view contains the reference tables.

When creating a variable on an attribute that is associated with one or several hierarchies, you can specify one of these hierarchies in the variable definition. With this option, the user can navigate the hierarchy, instead of a flat list, to select the values.

Input parameters support multiple values, which means that, at runtime, the end user has the possibility to provide several values to the parameter. Some examples of use cases include the following:

- Applying filters of type *List of values* and *Not in List*.
- Expression of calculated columns and Expression of filters in projection nodes, provided that the expression requires a multi-value input.

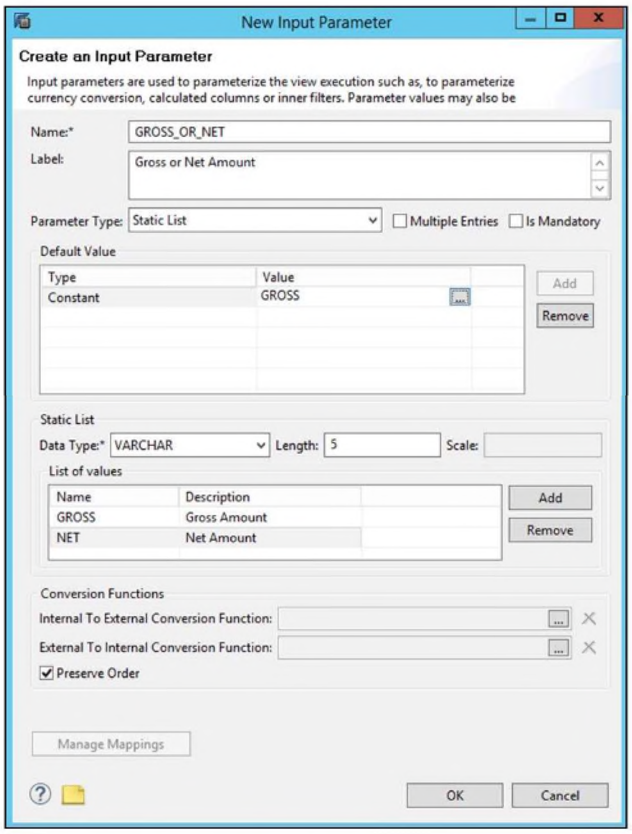
When you define an Input Parameter of type *Derived from Procedure/Scalar Function*, it is possible to map parameters to the input of the scalar function or procedure.

> Note:
> Input parameters of type *Derived from table* and *Derived from Procedure/Scalar Function* do not generate a prompt for the end user (they pass the parameter values directly), except if you select the *Input Enabled* option. In this case, the values returned by the table, procedure or scalar function, can be modified by the end user.

Creating Input Parameters

An input parameter can be used to determine which measure should be displayed in a particular column.

To illustrate, a calculated column *AMOUNT* can contain either the gross amount or the net amount, depending on the value that the end user chooses when querying the view.



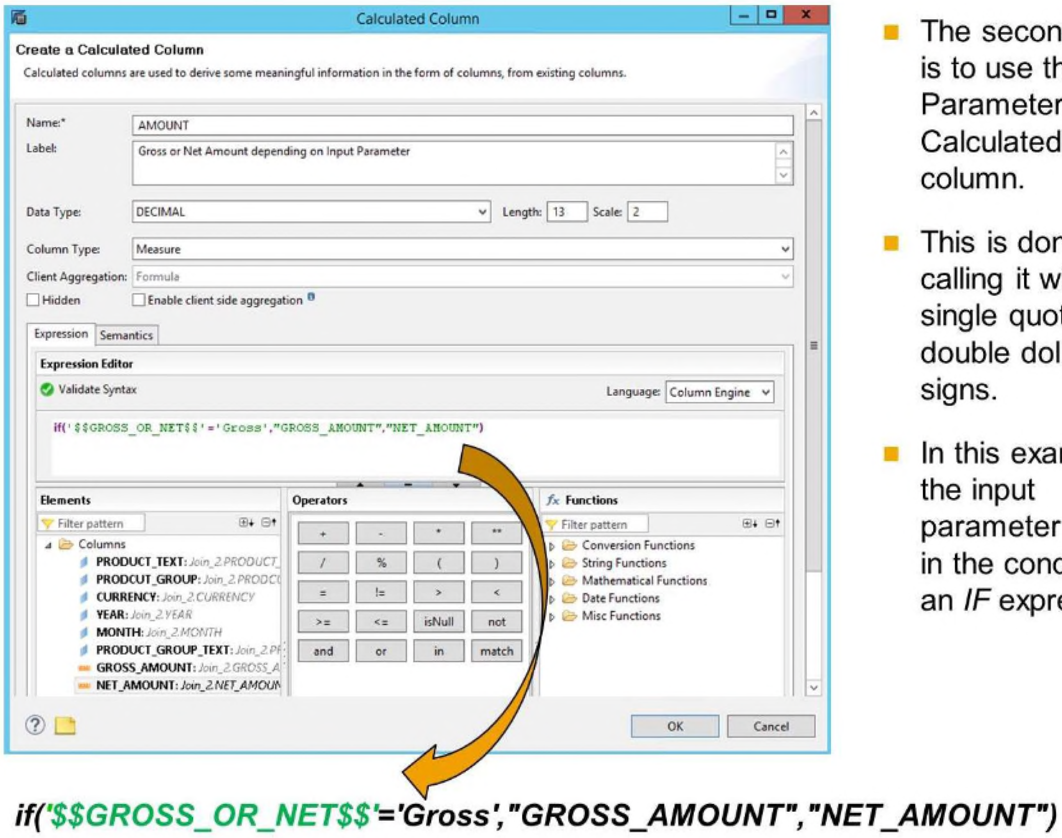
- If we want the end user to decide whether Gross or Net amount should be shown in a view, the first step is to create an input parameter that will be used in a calculation.
- The Input Parameter can be of any suitable type, for example a Static List type.
- In this example, the user will be able to choose either "Gross Amount" or "Net Amount".
- Default value GROSS will be assigned to the input parameter if the user does not specify anything.

Figure 144: Creating Input Parameters

An input parameter used within a formula does not necessarily have to be of type *Static List*. For example, it can also be a *Direct* numeric value used in multiplication or any other calculation type.

> Note:
> Since SAP HANA SPS10, Variables and Input Parameter now support multiple default values.

Calling an Input Parameter in a Calculation



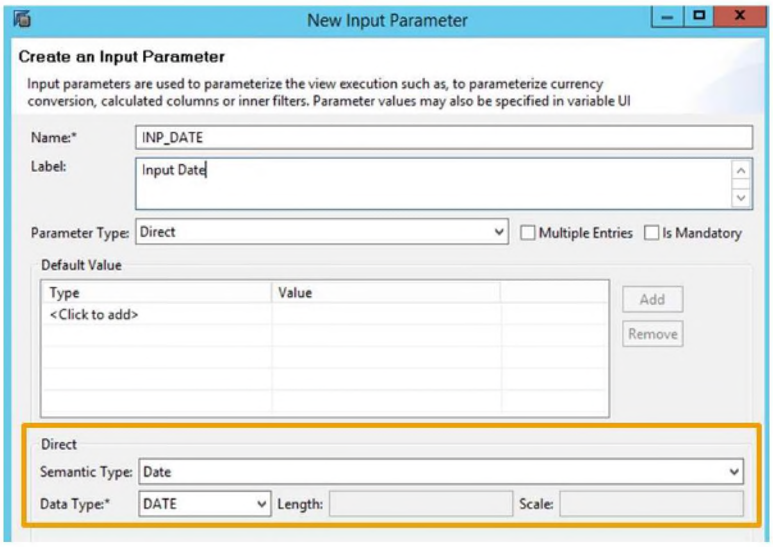
- The second step is to use the Input Parameter in a Calculated column.
- This is done by calling it within single quotes and double dollar signs.
- In this example, the input parameter is used in the condition of an *IF* expression

Figure 145: Calling an Input Parameter in a Calculation

In the example in the figure, Calling an Input Parameter in a Calculation, if the user selects *GROSS*, the calculated column (of type *Measure*) will display the GROSS_AMOUNT measure in the *AMOUNT* column. Any other selection will result in NET_AMOUNT.

> Note:
> This example shows that, in some cases, it is useful to define the list of values carefully (for example, with a static list to avoid unexpected results).

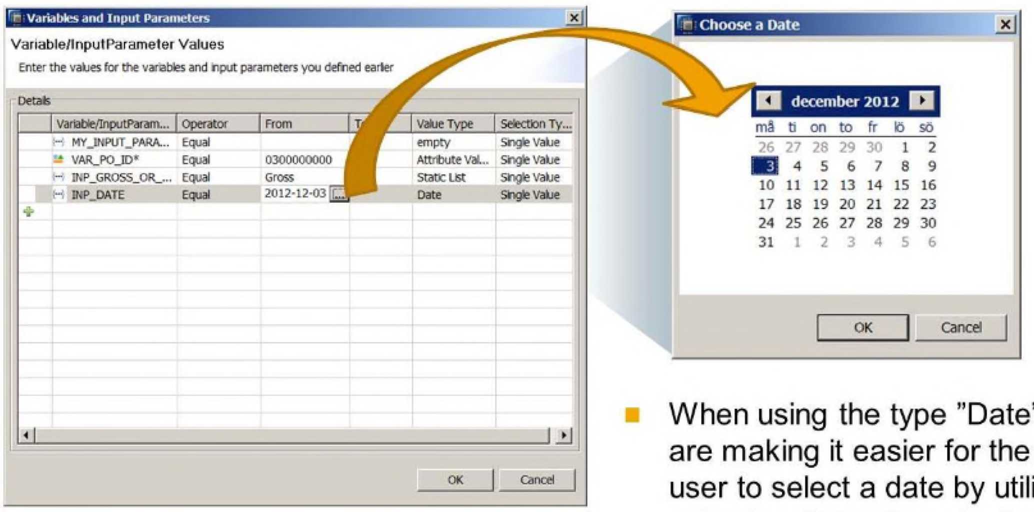
Input Parameter Using Dates



- An Input Parameter type of type “Direct” with a semantic type “Date” can be useful when you want to create calculations based on a date specified by the reporting user.
- You can create a date range by creating a pair of input parameters (for example, "Date From" and "Date To")
- Note that the Data Type must be set to "DATE".

Figure 146: Input Parameter Using Dates

Using a Calendar Dialog for Date Input Parameters



- When using the type ”Date” you are making it easier for the end user to select a date by utilizing a calendar dialog for selecting the appropriate date.

Figure 147: Using a Calendar Dialog for Date Input Parameters

In the figure, Using a Calendar Dialog for Date Input Parameters, the user is asked for a Single Value. Dates can also be selected as ranges.

Passing Parameters to External Views for Value Help

- Variables and Input Parameter can be mapped to variables and input parameters from external views
  - Allows filtering and customizing value help lists from external views
  - Supported with Analytic and Calculation Views (Graphical and Script)
- A Manage Mapping dialog box can be opened from:
  - The variable/input parameter creation dialog box
  - The "Semantics" node

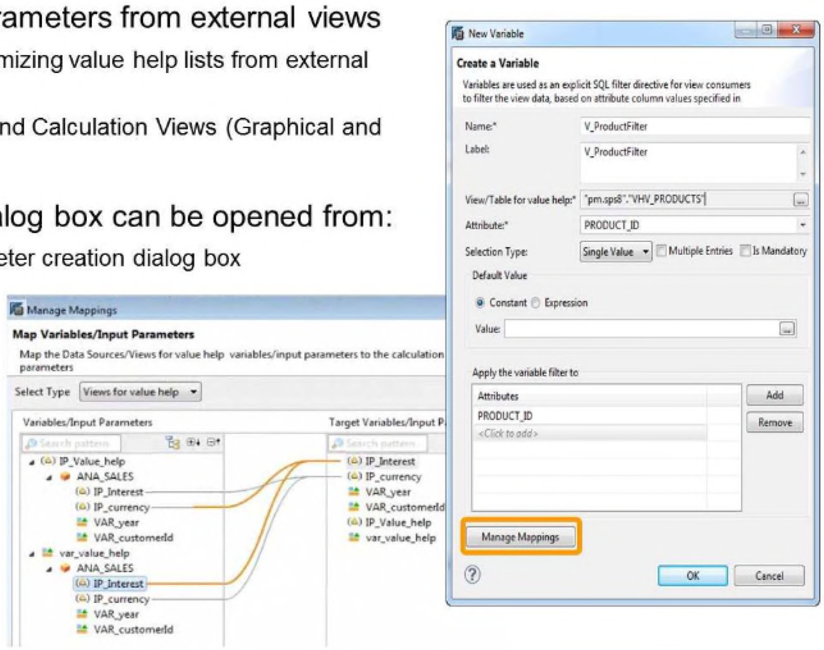


Figure 148: Passing Parameters to External Views for Value Help

When you execute a Calculation View in which variables or input parameters are defined, it is possible to pass the value of these variables/input parameters (entered by the end-user at runtime) to the source information views (for example, the Calculation View used in a projection or aggregation node of another Calculation View).

To enable this, you must map the variables or input parameters from the source information views to the ones defined in the Calculation View.

Unit 2
Exercise 12

Use Variables and Input Parameters

Exercise Objectives

After completing this exercise, you will be able to:

- Create variables
- Create input parameters with direct input and static value list
- Use input parameters in calculated columns

Business Example

You are a consultant at a customer that sells electronics. You have been asked to build a calculation view to analyze sales data for certain customers. Two marketing e-mails have been sent out to each customer. Your users want to find out how many days have passed between the two mailing campaigns and the order dates.

Task 1: Create a New Calculation View for the Marketing E-mail Campaign Analysis

1. Close all open views.

2. Create a new calculation view using the data in the following table:

Field	Value
Name:	CV_CAMPAIGN_ANALYSIS_##
Label:	Email Campaign Analysis
Package:	STUDENT##
View Type:	Calculation View
Subtype:	Standard
Type:	Graphical
Data Category:	CUBE

3. In the *Calculation View* scenario, under the *Aggregation* node, add a *Join* node.

4. Add the following tables to the Join node:

   - TRAINING.CUSTOMER
   - TRAINING.SALES_DATA

5. In your Join node, create a join between the tables:

Field	Value
Left Table	TRAINING.CUSTOMER

Field	Value
Right Table	TRAINING.SALES_DATA
Join Columns	CUSTOMER_ID = CUSTOMER_ID
Join Type	Left Outer
Cardinality	1..n

6. In the Join node, add the following columns to output:
   - CUSTOMER.CUSTOMER_TEXT
   - SALES_DATA.CURRENCY
   - SALES_DATA.AMOUNT
7. In the Scenario, connect the Join node to the *Aggregation* node.
8. In the *Aggregation* node, add the following columns:

Column Name	Action
CUSTOMER_TEXT	Add To Output
AMOUNT	Add As Aggregated Column
CURRENCY	Add To Output

9. In the *Semantics* node, rename the columns as per below table:

Column	New Name	New Label
CUSTOMER_TEXT	CUSTOMER	CUSTOMER
AMOUNT	SALES_AMOUNT	SALES_AMOUNT

10. In the *Semantics* node, Auto-Assign columns as measures and attributes.

Task 2: Create a Variable to Enable Customer Selection and Filtering

1. In the semantics, create a variable using the following details:

Field	Value
Name	VAR_CUSTOMER
Label	Customer Selector
Attribute	CUSTOMER
Selection Type	Single Value
Multiple Entries	[Selected]
Is Mandatory	[Selected]

2. Save and activate the view.

> Note:
> An intermediate solution object is available at this stage:
>
> *STUDENT00 → ha300 → CV_CAMPAIGN_ANALYSIS_00_STAGE1*

3. Display the results as Raw Data.
4. Select the customers *Becker Berlin* and *High Tech Park*.

> Hint:
> To set the value of a variable or input parameter, choose in the corresponding *From* cell, and then the ellipsis (...) button.
>
> If needed, choose the + sign to add another entry.

Task 3: Delete the Variable CUSTOMER and Add a New Column from the Source Table to the Output

The new columns will be used later on to calculate the number of days elapsed since the last marketing campaigns.

1. Delete the variable *VAR_CUSTOMER*, which will no longer be used because you want to visualize data for all the customers.

> Note:
> A dialog box indicates that this will also remove references to the variable.

2. In the *Join* node, add the column *SALES_DATA.SQL_DATE* to the output.
3. Add the new column to the *Aggregation* node.
4. In the output of the *Aggregation* node, rename the column *SQL_DATE* as **`SALE_DATE`**.
5. In the *Semantics*, define the new column as type *Attribute*.

Task 4: Create an Input Parameter to Select the E-mail Date, and a Calculated Column to Determine the Number of Days between the E-mail and the Sale Dates

Calculate the number of days elapsed between the e-mail marketing campaign and the sales orders.

> Note:
> There were two marketing campaigns in 2012, on January 25 and February 23.

1. In the semantics, create a new input parameter using the following details:

Hint:
To create an input parameter, select the *Parameters/Variables* tab of the *Semantics* node, choose the arrow next to the + sign and choose *Create Input Parameter*.

Field	Value
Name	INP_EMAIL_DATE
Label	The date when the e-mail was sent
Parameter Type	Static List
Multiple Entries	[not selected]
Is Mandatory	[selected]
Default value	[none]
Data Type	Date
List of values	• *Name:* 2012-01-25 *Description:* First e-mail • *Name:* 2012-02-23 *Description:* Second e-mail

Hint:
Use *Add* to define the list of values.

2. In the Aggregation node, add a calculated column to determine the number of days elapsed between the e-mail and the sale dates, using the following details:

Field	Value
Name	**DAYS_ELAPSED**
Description	**Days between Sale Date and e-mail**
Date Type	*INTEGER*
Column Type	*Attribute*
Expression	`daysbetween("SALE_DATE", date('$$INP_EMAIL_DATE$$'))`

Note:
The expression may not validate.

3. Save and activate the Calculation view.

> Note:
> An intermediate solution object is available at this stage:
> *STUDENT00 → ha300 → CV_CAMPAIGN_ANALYSIS_00_STAGE2*

4. Preview the data, and select a date for the parameter *INP_EMAIL_DATE*, for example 2012-01-25.

In the displayed data, check that the number of elapsed days is correctly calculated based on the input parameter you selected.

Task 5: Modify the Input Parameter so that the User can Enter Other Dates than the ones Provided by a Static List

When you query the view, the input parameter should be set by default to the current date.

1. Edit the Input Parameter *INP_EMAIL_DATE* to change its parameter according to the following table:

Field	Value
Parameter Type	Direct
Multiple Entries	[not selected]
Is Mandatory	[selected]
Default Value	Expression: **`now()`**
Semantic Type	Date
Data Type	DATE

2. Save and activate the Calculation view.

3. Display the results as Raw Data.

4. In the *Variables and Input Parameters* screen, select a date using the calendar, for example *2013-01-01*, or keep the default value (the current date).

5. In the displayed data, check that the number of elapsed days is correctly calculated based on the input parameter you defined.

Unit 2
Solution 12

Use Variables and Input Parameters

Exercise Objectives

After completing this exercise, you will be able to:

- Create variables
- Create input parameters with direct input and static value list
- Use input parameters in calculated columns

Business Example

You are a consultant at a customer that sells electronics. You have been asked to build a calculation view to analyze sales data for certain customers. Two marketing e-mails have been sent out to each customer. Your users want to find out how many days have passed between the two mailing campaigns and the order dates.

Task 1: Create a New Calculation View for the Marketing E-mail Campaign Analysis

1. Close all open views.
2. Create a new calculation view using the data in the following table:

Field	Value
Name:	CV_CAMPAIGN_ANALYSIS_##
Label:	Email Campaign Analysis
Package:	STUDENT##
View Type:	Calculation View
Subtype:	Standard
Type:	Graphical
Data Category:	CUBE

   a) Select *New* → *Calculation View* from the context menu of your package in the *Content* node.

   b) Enter the view properties as shown in the table.

   c) Choose *Finish*.

3. In the *Calculation View* scenario, under the *Aggregation* node, add a *Join* node.

   a) In the *Scenario* Palette, choose the *Join* node.

   b) In the *Scenario*, place a new Join node under the *Aggregation* node.

4. Add the following tables to the Join node:

- TRAINING.CUSTOMER
- TRAINING.SALES_DATA

a) Drag the above tables to the Join node from the TRAINING schema of the *Catalog* node.

5. In your Join node, create a join between the tables:

Field	Value
Left Table	TRAINING.CUSTOMER
Right Table	TRAINING.SALES_DATA
Join Columns	CUSTOMER_ID = CUSTOMER_ID
Join Type	Left Outer
Cardinality	1..n

a) Drag a connecting line between the columns *TRAINING.CUSTOMER.CUSTOMER_ID* and *TRAINING.SALES_DATA.CUSTOMER_ID*.

b) Double-click the join connector and define the properties as per above table.

6. In the Join node, add the following columns to output:

- CUSTOMER.CUSTOMER_TEXT
- SALES_DATA.CURRENCY
- SALES_DATA.AMOUNT

a) In the Join node, right-click the above columns and select *Add To Output* from the context menu

7. In the Scenario, connect the Join node to the *Aggregation* node.

a) In the Scenario of the calculation view, drag a connecting line from the Join node to the *Aggregation* node.

8. In the *Aggregation* node, add the following columns:

Column Name	Action
CUSTOMER_TEXT	Add To Output
AMOUNT	Add As Aggregated Column
CURRENCY	Add To Output

a) In the *Aggregation* node, right click each column and select *Add To Output/Add As Aggregated Column* as shown in the table.

9. In the *Semantics* node, rename the columns as per below table:

Column	New Name	New Label
CUSTOMER_TEXT	CUSTOMER	CUSTOMER

AMOUNT	SALES_AMOUNT	SALES_AMOUNT

a) Select the *Semantics* node.

b) In the *Columns* tab, define the new names and labels as specified in the table.

10. In the *Semantics* node, Auto-Assign columns as measures and attributes.

a) In the *Columns* tab of the *Semantics* node, choose *Auto-Assign*.

Task 2: Create a Variable to Enable Customer Selection and Filtering

1. In the semantics, create a variable using the following details:

Field	Value
Name	VAR_CUSTOMER
Label	Customer Selector
Attribute	CUSTOMER
Selection Type	Single Value
Multiple Entries	[Selected]
Is Mandatory	[Selected]

a) In the *Semantics* node, select the *Parameters/Variables* tab.

b) Choose +.

c) Enter the details as shown in the table.

d) Choose *OK*.

2. Save and activate the view.

> Note:
> An intermediate solution object is available at this stage:
> *STUDENT00 → ha300 → CV_CAMPAIGN_ANALYSIS_00_STAGE1*

a) Choose *Save and Activate*.

3. Display the results as Raw Data.

a) Choose *Data Preview*.

b) Select the *Raw Data* tab.

4. Select the customers *Becker Berlin* and *High Tech Park*.

> Hint:
> To set the value of a variable or input parameter, choose in the corresponding *From* cell, and then the ellipsis (...) button.
> If needed, choose the + sign to add another entry.

a) In the *Variables and Input Parameters* screen, for the *VAR_CUSTOMER* variable, choose *From* and then choose the ellipsis (...) button.

b) From the list, select *Becker Berlin*.

c) Choose *OK*.

d) Choose + to add another entry and repeat the previous steps for customer *High Tech Park*.

e) Choose *OK*.

f) Select the *Raw Data* tab and check that the output includes the two specified customers only.

Task 3: Delete the Variable CUSTOMER and Add a New Column from the Source Table to the Output

The new columns will be used later on to calculate the number of days elapsed since the last marketing campaigns.

1. Delete the variable *VAR_CUSTOMER*, which will no longer be used because you want to visualize data for all the customers.

> Note:
> A dialog box indicates that this will also remove references to the variable.

a) In the *Parameters/Variables* of the *Semantics* node, select the variable *VAR_CUSTOMER*.

b) Choose the blue *X (delete)* button.

> Hint:
> You can also press **DEL**.

c) In the dialog box, choose *Yes*.

d) In the *Impact* dialog box, choose *OK*.

2. In the *Join* node, add the column *SALES_DATA.SQL_DATE* to the output.

a) In the Scenario, select the *Join* node.

b) Right-click the column *SALES_DATA.SQL_DATE* and select *Add to Output* from the context menu.

3. Add the new column to the *Aggregation* node.

a) In the *Scenario*, select the *Aggregation* node.

b) Right-click the column *SQL_DATE* and select *Add to Output* from the context menu.

4. In the output of the *Aggregation* node, rename the column *SQL_DATE* as **SALE_DATE**.

a) In the *Columns* tab of the Aggregation node, change the name and label of the column *SQL_DATE* to **SALE_DATE**.

5. In the *Semantics*, define the new column as type *Attribute*.

a) In the *Scenario*, select the *Semantics* node.

b) In the *Columns* tab, select the *Type* cell for the *SALE_DATE* column and in the drop-down list, select *Attribute*.

Task 4: Create an Input Parameter to Select the E-mail Date, and a Calculated Column to Determine the Number of Days between the E-mail and the Sale Dates

Calculate the number of days elapsed between the e-mail marketing campaign and the sales orders.

Note:
There were two marketing campaigns in 2012, on January 25 and February 23.

1. In the semantics, create a new input parameter using the following details:

Hint:
To create an input parameter, select the *Parameters/Variables* tab of the *Semantics* node, choose the arrow next to the + sign and choose *Create Input Parameter*.

Field	Value
Name	INP_EMAIL_DATE
Label	The date when the e-mail was sent
Parameter Type	Static List
Multiple Entries	[not selected]
Is Mandatory	[selected]
Default value	[none]
Data Type	Date
List of values	• *Name:* 2012-01-25 *Description:* First e-mail • *Name:* 2012-02-23 *Description:* Second e-mail

Hint:
Use *Add* to define the list of values.

a) In the *Parameters/Variables* tab of the *Semantics* node, choose the arrow next to the + sign and choose *Create Input Parameter*.

b) Enter the details as shown in the table.

c) Choose *OK*.

2. In the Aggregation node, add a calculated column to determine the number of days elapsed between the e-mail and the sale dates, using the following details:

Field	Value
Name	**DAYS_ELAPSED**
Description	**Days between Sale Date and e-mail**
Date Type	*INTEGER*
Column Type	*Attribute*
Expression	`daysbetween("SALE_DATE", date('$$INP_EMAIL_DATE$$'))`

> Note:
> The expression may not validate.

a) In the *Output* pane of the *Aggregation* node, right-click the *Calculated Columns* folder and choose *New Calculated Column*.

b) Enter the details as shown in the table.

c) Choose *OK*.

3. Save and activate the Calculation view.

> Note:
> An intermediate solution object is available at this stage:
>
> *STUDENT00 → ha300 → CV_CAMPAIGN_ANALYSIS_00_STAGE2*

a) Choose *Save and Activate*.

4. Preview the data, and select a date for the parameter *INP_EMAIL_DATE*, for example 2012-01-25.

a) Choose the *Data Preview* button.

b) For your input parameter *INP_EMAIL_DATE*, choose in the *From* cell and then the ellipsis (...) button.

c) Select one of the dates, for example 2012-01-15.

d) Choose *OK* twice.

e) Select the *Raw Data* tab.

In the displayed data, check that the number of elapsed days is correctly calculated based on the input parameter you selected.

Task 5: Modify the Input Parameter so that the User can Enter Other Dates than the ones Provided by a Static List

When you query the view, the input parameter should be set by default to the current date.

1. Edit the Input Parameter *INP_EMAIL_DATE* to change its parameter according to the following table:

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

Field	Value
Parameter Type	Direct
Multiple Entries	[not selected]
Is Mandatory	[selected]
Default Value	Expression: **now()**
Semantic Type	Date
Data Type	DATE

a) In the *Semantics* node, select the *Parameter/Variable* tab. In *Variables/Input Parameters*, double-click the input parameter *INP_EMAIL_DATE*.

b) Set the values using the data in the table.

c) To define the default value, select the *Expression* checkbox and choose the ellipsis (...) button.

d) In the *Expression Editor* screen, expand the *Date Functions* folder and drag the *now()* function to the *Expression Editor* pane. Then, validate the syntax and choose *OK*.

e) Choose *OK*.

2. Save and activate the Calculation view.

   a) Choose *Save and Activate*.

3. Display the results as Raw Data.

   a) Choose *Data Preview*.

   b) Select the *Raw Data* tab.

4. In the *Variables and Input Parameters* screen, select a date using the calendar, for example *2013-01-01*, or keep the default value (the current date).

   a) For your input parameter *INP_EMAIL_DATE*, choose the *From* cell, and then the ellipsis (...) button.

   b) If you do not want to keep the current date, select a date using the calendar, for example **2013-01-01** and choose OK.

   c) Choose *OK*.

   d) Select the *Raw Data* tab.

5. In the displayed data, check that the number of elapsed days is correctly calculated based on the input parameter you defined.

LESSON SUMMARY

You should now be able to:

- Use variables and input parameters
- Create variables and use them to filter data
- Create input parameters

Unit 2
Lesson 5

Implementing Currency Conversion

LESSON OVERVIEW

This lesson describes how to set up currency conversion for measures.

Business Example

For a worldwide company, sales are conducted in a lot of different currencies. However, you want to display data and amounts with one single currency to be able to aggregate data. So you need to set up a currency conversion.

You want to know more about the native functionality of SAP HANA to implement currency conversion in your information models.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Explain the general principles of currency conversion
- Apply currency conversion in Calculation Views

Currency Conversion in Data Reporting Rationale

- As most front-end tools do not allow defining or switching reporting currency in the UI, and as there might not be such information in master data, we have to convert the possibly many monetary document currencies into just a few.
- SAP HANA has the necessary functions needed to achieve currency conversion during data modeling.

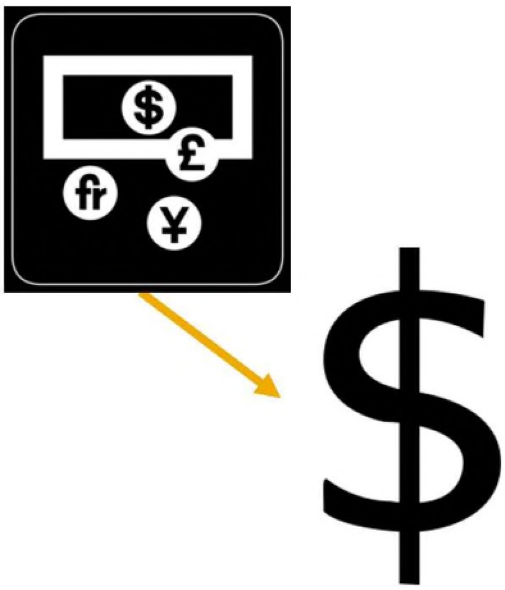


Figure 149: Currency Conversion

When you build information models, the source data is often expressed in a single currency. Typically, for sales data, this would be the transaction currency at the date of sale.

For reporting purposes, however, it is very often necessary to convert the currencies. The purposes can include (but are not limited to) the following:

- Corporate Reporting

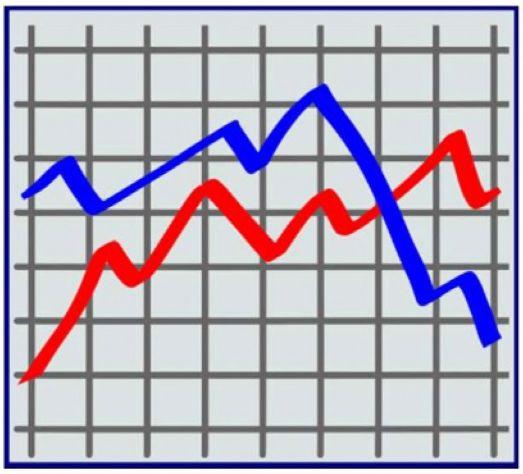
  When using a global reporting currency in corporate reporting, all values should be displayed in the global reporting currency.
- Regional Reporting

  For example, the US region might want to see European figures in USD.
- P&L Reporting

  You might want to analyze the effects of currency gains and losses.
- Accounting

  Data conversion is a strong requirement in multi-currency general ledger transactions.

Currency Conversion



- As currency exchange rates fluctuate constantly in the global markets, when converting it is necessary not only to define the **source** and **target** currencies when converting, but also to define the **time** when currency conversion should take place.
- Examples could be:
  - Billing Date
  - Posting Date
  - Financial Year End
  - Today's Date

Figure 150: Currency Conversion

Due to the permanent fluctuation of currency exchange rates, you have to define a smart conversion process and, in particular, define which date must be considered to define the conversion rate to apply.

This requirement is even stronger when the source or target currency is volatile.

Native Currency Conversion in SAP HANA Information Models

SAP HANA offers an elaborate conversion mechanism that is based on the following building blocks:

- A set of technical tables to store master data about currencies, exchange rate types, and the exchange rate values.
- The concept of Semantic Type, which allows a flag to measure with the *Amount with Currency* semantic type.
- An interface to define, for each amount measure, how the conversion should be processed by the information model (which rate and conversion date(s) should be applied, where to find the source and target currency.

Currency Conversion Approaches in SAP HANA

In SAP HANA, data conversion can be implemented in both graphical and script based views.

- Graphical calculation views in SAP HANA provide the easiest way to convert currencies, as the conversion modeling can be done using the graphical interface.
- Alternatively, in case of constraints in the master data or because of the complexity of the reporting requirements, you can model the currency conversion within a scripted view.

  However, this feature is based on a SQLScript function, *CONVERT_CURRENCY*, which is based on Column Engine Plan Operators (or CE Functions), which are now deprecated.

> Note:
> If you have to adapt an existing implementation of this *CONVERT_CURRENCY* function, you can find more information in the *SAP HANA SQL and System Views Reference* guide, available at http://help.sap.com/hana.

Applying Conversion to Lower Aggregation Nodes

From SAP HANA SPS12 onwards, it is possible to define a currency conversion on any aggregation node of a calculation view. This enhancement allows you to maintain a currency conversion at an intermediate level of your calculation view. For example, when you need to combine two different data sets with a Union node, and only one of the data sets needs to be converted.

> Note:
> Up to SPS11, currency conversion could only be defined at the upmost level of the calculation view, either the upper Aggregation node of a Cube Calculation View, or the Star Join node of a Cube with Star Join Calculation View.

TCUR Schema

One of the key building blocks to enable data conversion in SAP HANA is a set of tables to define currencies, exchange rate types, and to store the conversion rates.

Table 8: Required Conversion Tables

Table Name	Description
TCURC	Currency codes
TCURR	Exchange rates
TCURV	Exchange rate types for currency translation
TCURF	Conversion factors
TCURN	Quotations
TCURX	Decimal places in currencies

These tables exist in most SAP Systems (in particular, SAP Business Suite and SAP Business Warehouse).

To enable conversion in SAP HANA, these tables must be available in the SAP HANA database, for example in a dedicated schema for currencies and exchange rates.

If the models are based on data replicated from another system, the replication process should also include these tables so that they are always synchronized with the ones located on the source system.

Implementing Conversion in Calculation Views

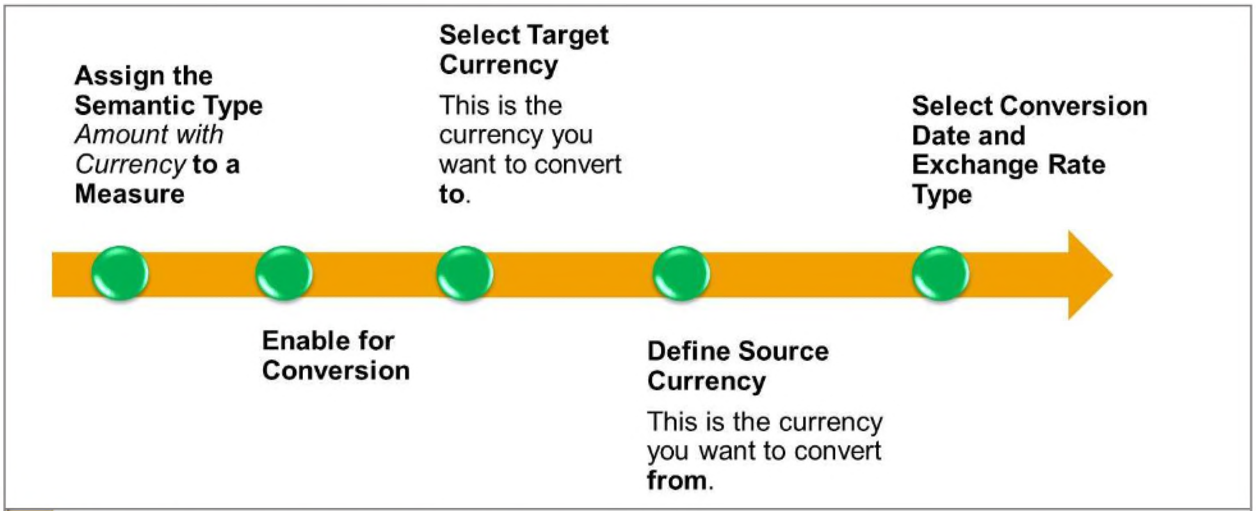


Figure 151: Currency Conversion Process Flow

The figure, Currency Conversion Process Flow, depicts the process flow for currency conversion within a *Calculation View*.

Setting the Semantic Type

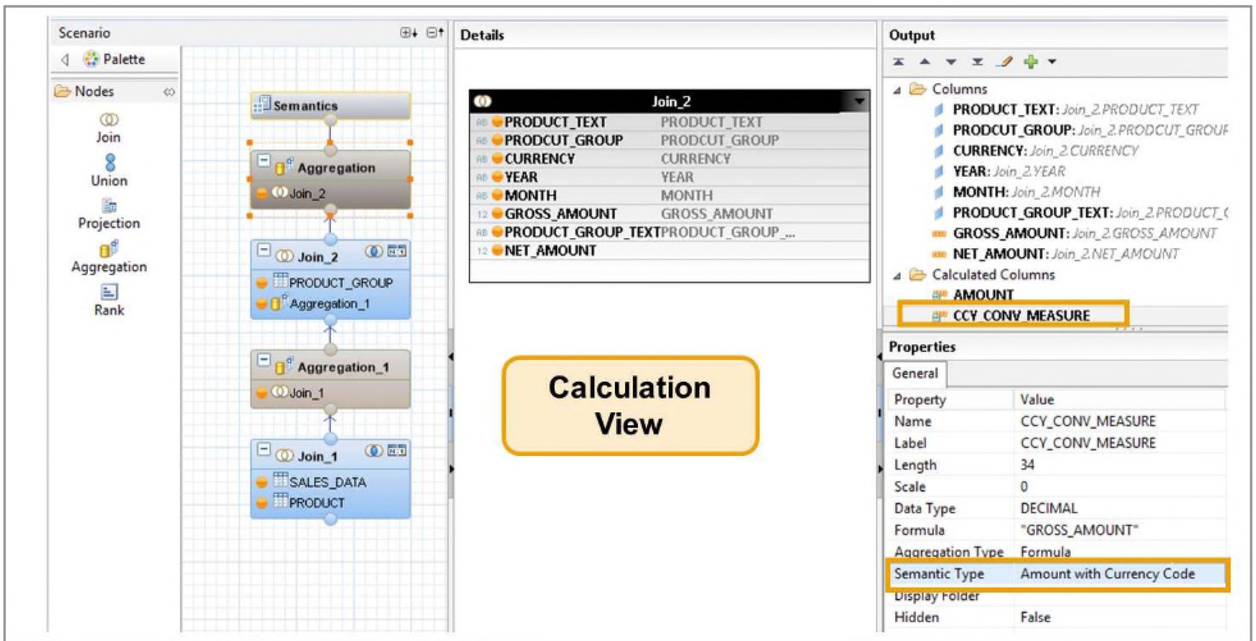


Figure 152: Setting the Semantic Type

By default, a measure has no Semantic Type.

When a measure contains an amount, and if you want to enable conversion, you need to change its Semantic Type to *Amount with Currency Code*. You can do that in the following ways:

How to Assign the Semantic Type?

- In the *Output* pane of the Aggregation node, select an aggregated column and, in the *General Properties* pane, define the Semantic Type. You can also double-click the column.

  This can also be done on the Star Join node of a Cube with Star Join calculation view.

- Alternatively, when the currency conversion is implemented at the upmost level of the calculation view, select the Semantics node, and choose *Assign Semantics*.

> Note:
> The Semantic Type *Amount with Currency Code* can be used even when conversion is not actually used.
>
> In this case, you only have to set the Currency property to identify the currency (or currencies) in which the amounts are expressed.

Defining Currency Conversion Settings

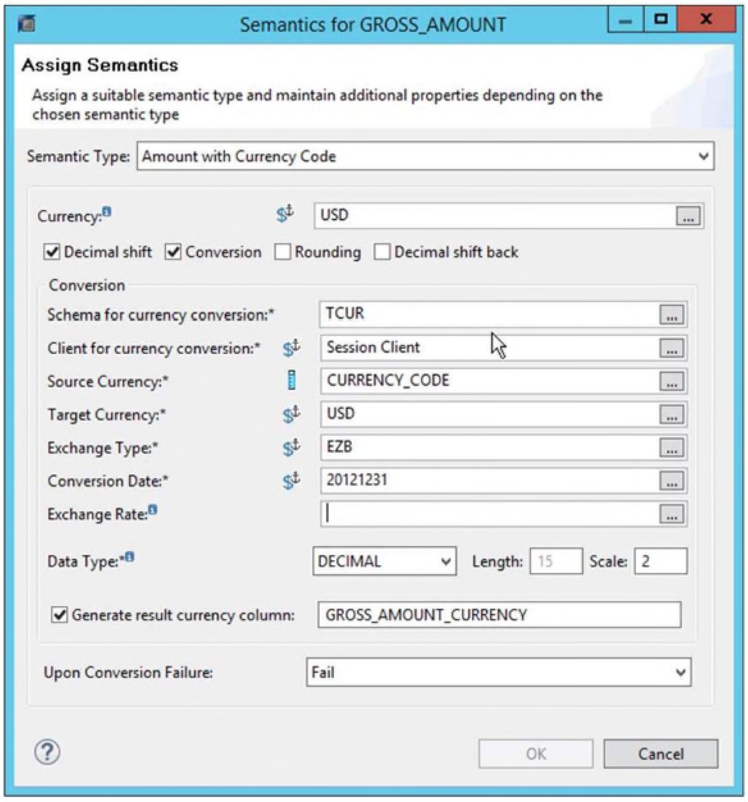


Figure 153: Currency Conversion Settings

After assigning a semantic type *Amount with Currency Code*, you can enable conversion and define the main parameters used for conversion.

Table 9: Key Settings for Currency Conversion

Setting	Description	Options
Schema for Currency Conversion	The schema that contains the *TCUR** tables	

Setting	Description	Options
Client for Currency Conversion	The client (MANDT) to use to filter the TCUR* tables content	*Session Client / Fixed* Client Number */ Column / Input parameter*
Source Currency	The currency in which the amounts to convert are expressed	*Fixed / Column*
Target Currency	The currency in which the amounts must be converted	*Fixed / Column / Input Parameter*
Exchange Type	The type of rate used to convert amounts. Example: Spot rate, average rate...	*Fixed / Column / Input Parameter*
Conversion Date	The date used to match an amount and the corresponding conversion rate	*Fixed / Column / Input Parameter*
Exchange Rate	(optional) A column from the source data that contains the exchange rate to be used	
Data Type	The data type of the converted measure	Example: Decimal (15,2)
Generate result currency column	If selected, this option creates a column that indicates for each converted amount the (target) currency in which it is expressed.	
Upon conversion failure	Specifies the behavior if the conversion cannot be executed (for example, if the rate	*Fail* (a query on the view generates an error), *NULL* (the column is not populated), *Ignore* (keeps the source amount without converting it)

It is important to carefully define how the exceptions must be handled when converting data. In addition, to reduce the risk of conversion failure, make sure the currency conversion tables TCUR* in your SAP HANA system are updated on a regular basis, in particular in a side-by-side scenario where they should always be in sync with the data imported from the remote SAP system.

Note:
The Result currency column is never exposed to client tools. It is only available to other SAP HANA views, where it can be used in additional calculations.

Decimal Shift and Rounding

By default, the precision of all values is 2 digits in SAP ERP tables.

As some currencies require accuracy in value, *Decimal Shift* moves the decimal points according to the settings in the TCURX currency table. If you want to round the result value after currency conversion to the number of digits of the target currency, select the *Rounding* checkbox.

Decimal shift back is necessary if the results of the calculation views are interpreted in ABAP. The ABAP layer, by default, always executes the decimal shift. In such cases, decimal shift back helps avoid wrong numbers due to a double shift.

Using an Input Parameter for the Currency

The *Input Parameter* can also be described as a prompt in that it asks the user what currency to use.

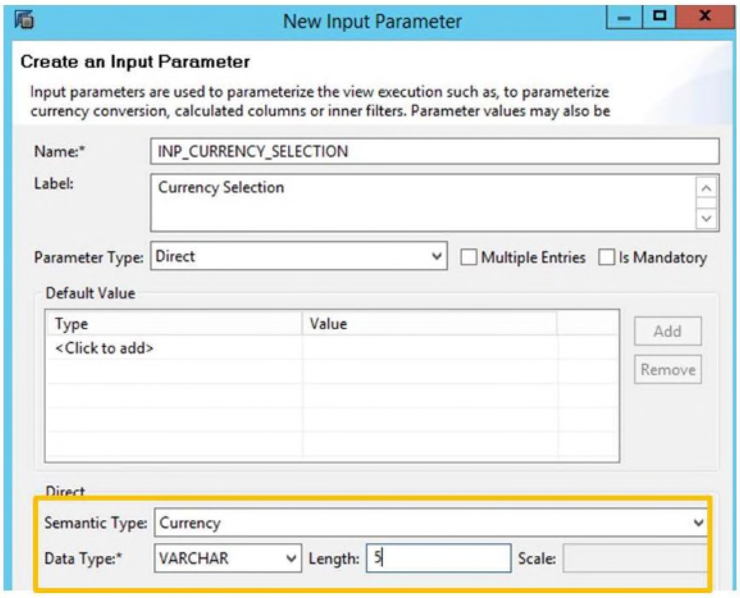


Figure 154: Creating an Input Parameter

If you want to define the currency at runtime, when the view is executed, you can create an input parameter.

VARCHAR (5) is the way the currency code is defined in the TCUR* tables, so to be consistent, it is recommended to define the input parameter with the same data type.

Unit 2
Exercise 13

Implement Currency Conversion in a Calculation View

Exercise Objectives

After completing this exercise, you will be able to:

- Apply Currency Conversion in a Calculation View
- Leverage Fixed Currencies

Business Example

You are at a customer site where EPM data is available. USD is the general corporate reporting currency.

You have been asked to build *Calculation Views* for SAP HANA for the purpose of displaying Sales Data.

Task 1: Create a Copy of an Existing Calculation View and Analyze its Definition and Data Preview

Use the new Calculation View later to implement data conversion.

1. Create a new Calculation View in your *STUDENT##* package, based on an existing one, with the following properties.

Field	Value
Name	**CVC_SO_CCY_##**
Label	**Sales Orders with Conversion**
Copy From	*training.ha300.CV_SO_CCY_TEMPLATE*

> Hint:
> You can use the copy and paste feature within the *Content* node of the *Systems* view.

2. Open the calculation view, and set the *Default Schema* to *TCUR*.

   This is necessary in order to define Semantics for conversion later on.

3. Preview the Data of the new Calculation View, and confirm data availability. In particular, pay attention to the *CURRENCY_CODE* column and examine the types of available currencies.

> Note:
> You must save, validate, and activate the new view before previewing the data.

Task 2: Implement Currency Conversion in the New Calculation View, with a Column-Based Currency Source and a Fixed Destination Currency

> Note:
> Use a conversion rate type called EZB (exchange rate provided by the European Central Bank). The training data includes conversion rates for EUR to USD and rate type EZB for two specific dates:
>
> - 31/12/2012: 1 EUR = 1.3203 USD
> - 31/12/2013: 1 EUR = 1.3744 USD

1. Modify the output columns of the Calculation View to have two amounts, one in the source currency, another in the conversion currency. To do so, add a second instance *GROSS_AMOUNT* as an Aggregated column to the output, and rename the two amount columns as follows:

Previous Name	New Name	New Label
GROSS_AMOUNT	*GROSS_AMOUNT_SRC*	*GROSS_AMOUNT_SRC*
GROSS_AMOUNT_1	*GROSS_AMOUNT_USD*	*GROSS_AMOUNT_USD*

Assign the type *Measure* to the new column.

> Note:
> As a first step, you will implement a conversion into a fixed currency: *USD*.

2. At this stage, save and activate the view, and preview the data.

   Note that the data is the same in both amount columns.

3. To enable the conversion, first assign the following semantics to the column *GROSS_AMOUNT_SRC*.

> Note:
> Do not enable this column for conversion, as we want to display it in the source currency.

Field	Value
Semantic	Amount with Currency Code

Field	Value
Currency	• Type: *Column* • Assigned Column: *CURRENCY_CODE*

4. Assign the following semantics to the column *GROSS_AMOUNT_USD*.

   You must define the semantics and enable conversion to the fixed currency *USD*.

Field	Value
Semantic Type	*Amount with Currency Code*
Currency	Type: *Fixed* Assigned currency: *USD*
Decimal shift	Yes
Conversion	Yes
With rounding	No
Decimal shift back	No
Schema for currency conversion	*TCUR*
Client for currency conversion	*Fixed* Client: **800**
Source Currency	Type: *Column* Assigned column: *CURRENCY_CODE*
Target Currency	Type: *Fixed* Assigned currency: *USD*
Exchange Type	*EZB*
Conversion Date	**20121231**
Exchange Rate	*[blank]*
Data Type	*Decimal (15,2)*
Generate result currency column	No
Upon Conversion Failure	*Fail*

5. Save and Activate the Calculation View, and preview the data.

   You will now see the two *Measures*, the *GROSS_AMOUNT_SRC* in source currency, as well as the *GROSS_AMOUNT_USD* converted in the corporate reporting currency (USD).

> Note:
> As the two measures are assigned a semantic *Amount with Currency Code*, in the data preview, the currency is displayed in the same column as the amount.

Task 3: Refine your Calculation View by using an Input Parameter for the Conversion Rate Date

The template Calculation View you used at the beginning of this exercise includes an Input Parameter that you have not used so far. You will now modify the Calculation view in order to modify and make use of this input parameter.

1. Modify the *INP_CONV_DATE* input parameter so that it suggests a fixed list of dates: *31/12/2012* and *31/12/2013*. These values must be created as follows:

   - Parameter Type: Static List
   - Static List Data Type: Date
   - Is Mandatory: Yes
   - List Of Values: (see below table)

Name	Description
20121231	**31/12/2012**
20131231	**31/12/2013**

   Entering a value for the input parameter must be mandatory when the view is executed.

   > Hint:
   > In the *Semantics* node, display the *Parameters/Variables* tab and double-click the *INP_CONV_DATE* object.

2. Modify the semantics assigned to the column *GROSS_AMOUNT_USD* so that the conversion date is not hard-coded in the calculation view, but based on the *INP_CONV_DATE* input parameter.

3. Save and Activate the view and preview the data.

   This time, the *Variables and Input Parameters* dialog box opens. You must set the conversion date value for the *INP_CONV_DATE* input parameter in the *From* column. Choose one of the two possible conversion dates.

   Execute a second data preview with the other date and check the data again to ensure the conversion amounts are different.

4. Close all the open tabs.

Unit 2
Solution 13

Implement Currency Conversion in a Calculation View

Exercise Objectives

After completing this exercise, you will be able to:

- Apply Currency Conversion in a Calculation View
- Leverage Fixed Currencies

Business Example

You are at a customer site where EPM data is available. USD is the general corporate reporting currency.

You have been asked to build *Calculation Views* for SAP HANA for the purpose of displaying Sales Data.

Task 1: Create a Copy of an Existing Calculation View and Analyze its Definition and Data Preview

Use the new Calculation View later to implement data conversion.

1. Create a new Calculation View in your *STUDENT##* package, based on an existing one, with the following properties.

Field	Value
Name	`CVC_SO_CCY_##`
Label	`Sales Orders with Conversion`
Copy From	*training.ha300.CV_SO_CCY_TEMPLATE*

> Hint:
> You can use the copy and paste feature within the *Content* node of the *Systems* view.

a) In the *Systems* View, select the Calculation View *training.ha300.CVC_SO_CCY_TEMPLATE*.

b) Copy the template (on your keyboard press **`Ctrl+C`**).

c) In the *Content* folder of the Catalog, select the destination folder *STUDENT##*.

d) Paste the template (on your keyboard press **`Ctrl+V`**) and name it as specified in the table.

2. Open the calculation view, and set the *Default Schema* to *TCUR*.

This is necessary in order to define Semantics for conversion later on.

a) Open the Calculation View.

b) In the *Semantics* node, select the *Properties* tab.

c) In the *Default Schema* dropdown list, choose *TCUR*.

3. Preview the Data of the new Calculation View, and confirm data availability. In particular, pay attention to the *CURRENCY_CODE* column and examine the types of available currencies.

> Note:
> You must save, validate, and activate the new view before previewing the data.

a) Choose *Save and Activate*.

b) After checking the Validation and Activation log, choose *Data Preview*.

c) Notice that all the data is expressed in EUR.

d) Close the Data Preview.

Task 2: Implement Currency Conversion in the New Calculation View, with a Column-Based Currency Source and a Fixed Destination Currency

> Note:
> Use a conversion rate type called EZB (exchange rate provided by the European Central Bank). The training data includes conversion rates for EUR to USD and rate type EZB for two specific dates:
>
> - 31/12/2012: 1 EUR = 1.3203 USD
> - 31/12/2013: 1 EUR = 1.3744 USD

1. Modify the output columns of the Calculation View to have two amounts, one in the source currency, another in the conversion currency. To do so, add a second instance *GROSS_AMOUNT* as an Aggregated column to the output, and rename the two amount columns as follows:

Previous Name	New Name	New Label
GROSS_AMOUNT	*GROSS_AMOUNT_SRC*	*GROSS_AMOUNT_SRC*
GROSS_AMOUNT_1	*GROSS_AMOUNT_USD*	*GROSS_AMOUNT_USD*

Assign the type *Measure* to the new column.

> Note:
> As a first step, you will implement a conversion into a fixed currency: *USD*.

a) In the *Aggregation* node, right-click the *GROSS_AMOUNT* column and choose *Add As Aggregated Column*.

b) Select the *Columns* tab of the *Semantics* node.

c) Enter the new names and labels for the two amount columns as shown in the table.

2. At this stage, save and activate the view, and preview the data.

   Note that the data is the same in both amount columns.

3. To enable the conversion, first assign the following semantics to the column *GROSS_AMOUNT_SRC*.

> Note:
> Do not enable this column for conversion, as we want to display it in the source currency.

Field	Value
Semantic	Amount with Currency Code
Currency	• Type: *Column* • Assigned Column: *CURRENCY_CODE*

a) In the *Semantics* node, select the *Columns* tab.

b) Select the *GROSS_AMOUNT_SRC* column and choose (*Assign Semantics*).

c) In the *Semantic Type* dropdown list, select *Amount with Currency Code*.

d) To assign the other settings as per the table above, choose the ellipsis *(...)* button at the right of the *Currency* field.

e) Choose *OK*.

4. Assign the following semantics to the column *GROSS_AMOUNT_USD*.

   You must define the semantics and enable conversion to the fixed currency *USD*.

Field	Value
Semantic Type	*Amount with Currency Code*
Currency	Type: *Fixed* Assigned currency: *USD*
Decimal shift	Yes
Conversion	Yes
With rounding	No
Decimal shift back	No
Schema for currency conversion	*TCUR*

Field	Value
Client for currency conversion	*Fixed* Client: **800**
Source Currency	Type: *Column* Assigned column: *CURRENCY_CODE*
Target Currency	Type: *Fixed* Assigned currency: *USD*
Exchange Type	*EZB*
Conversion Date	**20121231**
Exchange Rate	*[blank]*
Data Type	*Decimal (15,2)*
Generate result currency column	No
Upon Conversion Failure	*Fail*

a) In the *Semantics* node, select the *Columns* tab.

b) Select the *GROSS_AMOUNT_USD* column and choose (*Assign Semantics*).

c) Assign all the settings as specified in the table.

d) Choose *OK*.

5. Save and Activate the Calculation View, and preview the data.

   You will now see the two *Measures*, the *GROSS_AMOUNT_SRC* in source currency, as well as the *GROSS_AMOUNT_USD* converted in the corporate reporting currency (USD).

> Note:
> As the two measures are assigned a semantic *Amount with Currency Code*, in the data preview, the currency is displayed in the same column as the amount.

a) Choose *Save and Activate*.

b) Check the validation job.

c) Choose *Data Preview* and select the *Raw Data* tab.

Task 3: Refine your Calculation View by using an Input Parameter for the Conversion Rate Date

The template Calculation View you used at the beginning of this exercise includes an Input Parameter that you have not used so far. You will now modify the Calculation view in order to modify and make use of this input parameter.

1. Modify the *INP_CONV_DATE* input parameter so that it suggests a fixed list of dates: *31/12/2012* and *31/12/2013*. These values must be created as follows:

- Parameter Type: Static List
- Static List Data Type: Date
- Is Mandatory: Yes
- List Of Values: (see below table)

Name	Description
20121231	31/12/2012
20131231	31/12/2013

Entering a value for the input parameter must be mandatory when the view is executed.

Hint:
In the *Semantics* node, display the *Parameters/Variables* tab and double-click the *INP_CONV_DATE* object.

a) In the *Semantics* node, select the *Parameters/Variables* tab and double-click the *INP_CONV_DATE* input parameter.

b) Change the *Parameter Type* to *Static List*.

c) Select the *Is Mandatory* checkbox

d) For the *Data Type* of the Static List, choose *DATE*.

e) In the *List of values*, choose *Add* twice, and modify the two values as specified in the table.

f) Choose *OK*.

2. Modify the semantics assigned to the column *GROSS_AMOUNT_USD* so that the conversion date is not hard-coded in the calculation view, but based on the *INP_CONV_DATE* input parameter.

   a) In the *Columns* tab of the *Semantics* node, select the column GROSS_AMOUNT_USD and choose *(Assign Semantics)*.

   b) In the *Conversion Date* field, replace the fixed value *20121231* by Type: *Input Parameter* and selection: *INP_CONV_DATE*.

   c) Choose *OK*.

3. Save and Activate the view and preview the data.

   This time, the *Variables and Input Parameters* dialog box opens. You must set the conversion date value for the *INP_CONV_DATE* input parameter in the *From* column. Choose one of the two possible conversion dates.

   Execute a second data preview with the other date and check the data again to ensure the conversion amounts are different.

   a) Choose *Save and Activate*.

   b) Check the validation job.

c) Choose *Data Preview*.

d) In the *Variables and Input Parameters* screen, select the ellipsis *(...)* button in the *From* column, choose *20121231* and choose *OK* twice.

e) Select the *Raw Data* tab and check the results.

f) Preview the Calculation View again and repeat the previous steps to view the data with the other conversion date.

> Note:
> You can also refresh the Data Preview by choosing the *Refresh* button on the right.

4. Close all the open tabs.

a) Right-click any of the open tabs and choose *Close All*.

LESSON SUMMARY

You should now be able to:

- Explain the general principles of currency conversion
- Apply currency conversion in Calculation Views

Unit 2
Lesson 6

Creating Decision Tables

LESSON OVERVIEW

Decision Tables is a way of implementing Business Rules in SAP HANA. Having captured operational decisions the business rules can be turned into code using Decision Tables.

Business Example

As an insurance broker, you must react quickly to the market, so you want to have your business rules reflected quickly in your reports.

To facilitate this scenario you have decided to move your insurance Business Rules into Decision Tables.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create decision tables
- Consume decision tables in Calculation Views

Turning Business Rules into Decision Tables



- **Operational Decisions** can be codified using business rules management capability that is natively available in SAP HANA
- **Business Rules** are normally written on either master data (e.g. Products, Customers) or on transactional data (like buying patterns of customers in retail) or both together. When rules have to be run on "big data", it makes sense to run these rules closer to the data on top of which the rules are executed for performance and improved "time to insight".

Figure 155: Decision Tables Overview

Business Rules can be hidden interpreted deep in complex programming code, preventing transparency and agility.

Business Examples

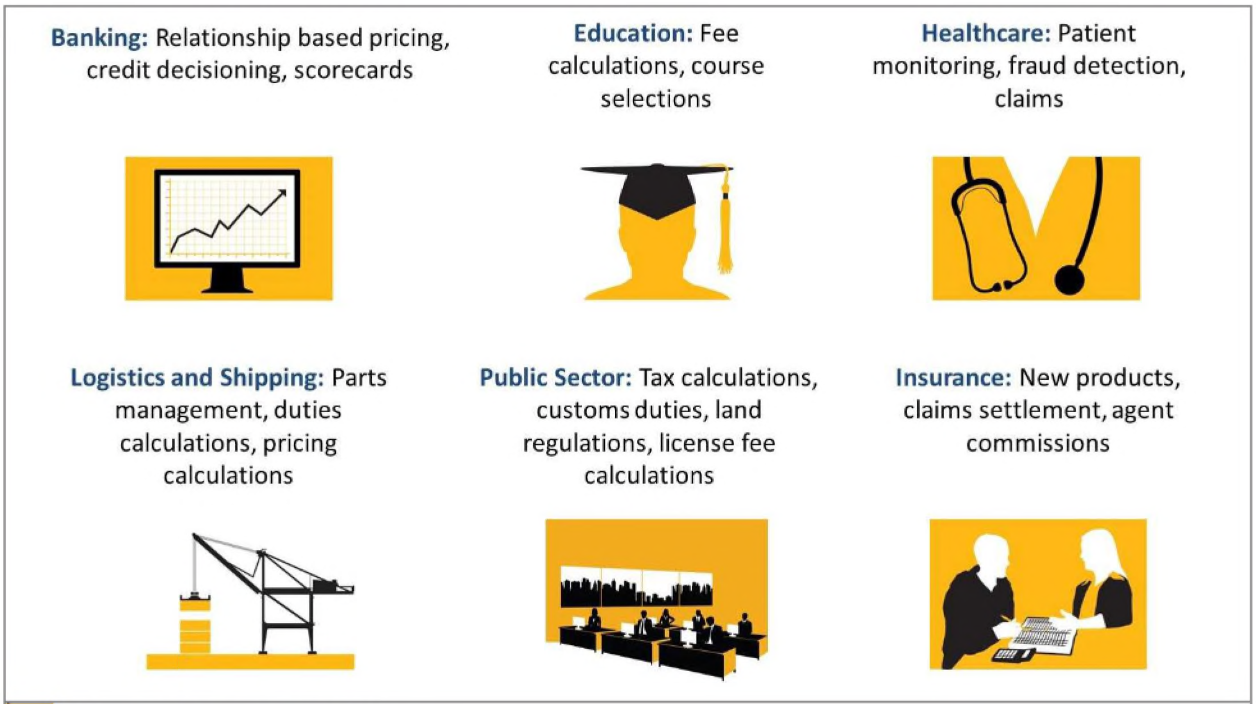


Figure 156: Business Examples

Helping Driving Agility

Consider for example a telecom company wanting to provide Best Tariffs Plans to customers based on their usage patterns when a customer calls their call center for a suggestion.

Call records represent a large volume dataset; business rules must suggest the best tariff plan for the customer without making the customer wait on the call for a long time.

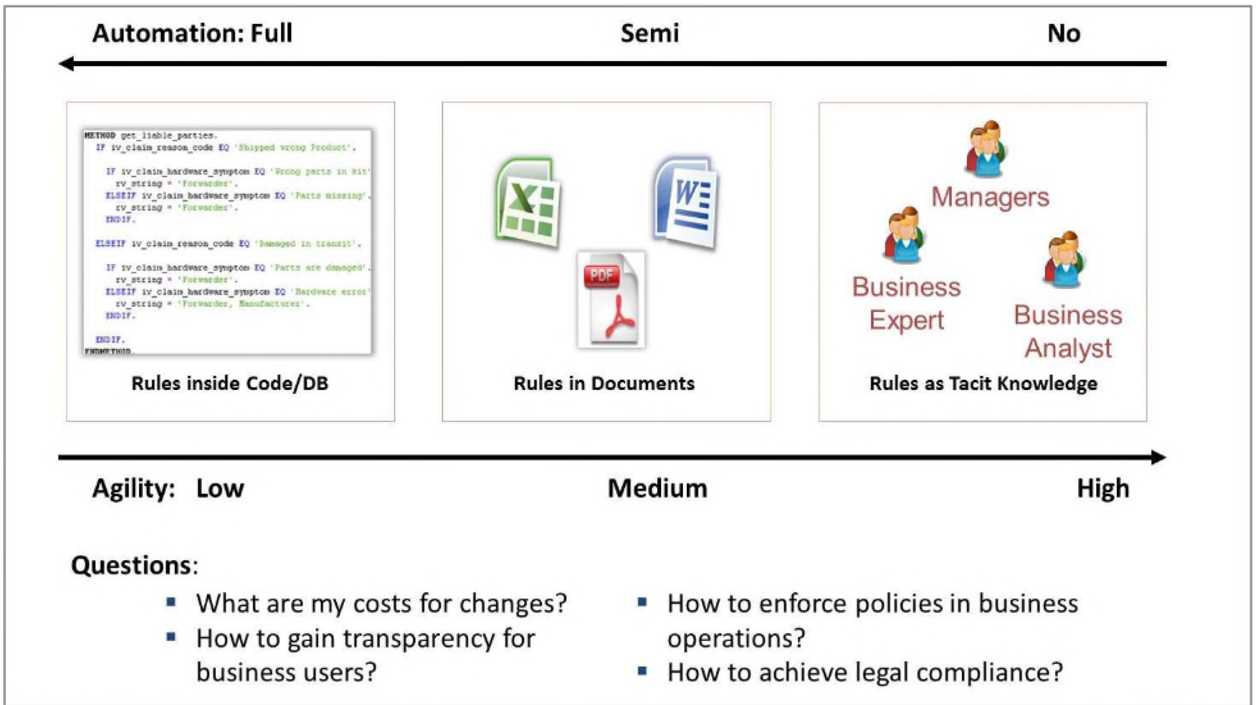


Figure 157: Helping Driving Agility

Different Users

Moving Business Rules into SAP HANA and Decision Tables will help drive business agility as well as offer transparency to Business Users.

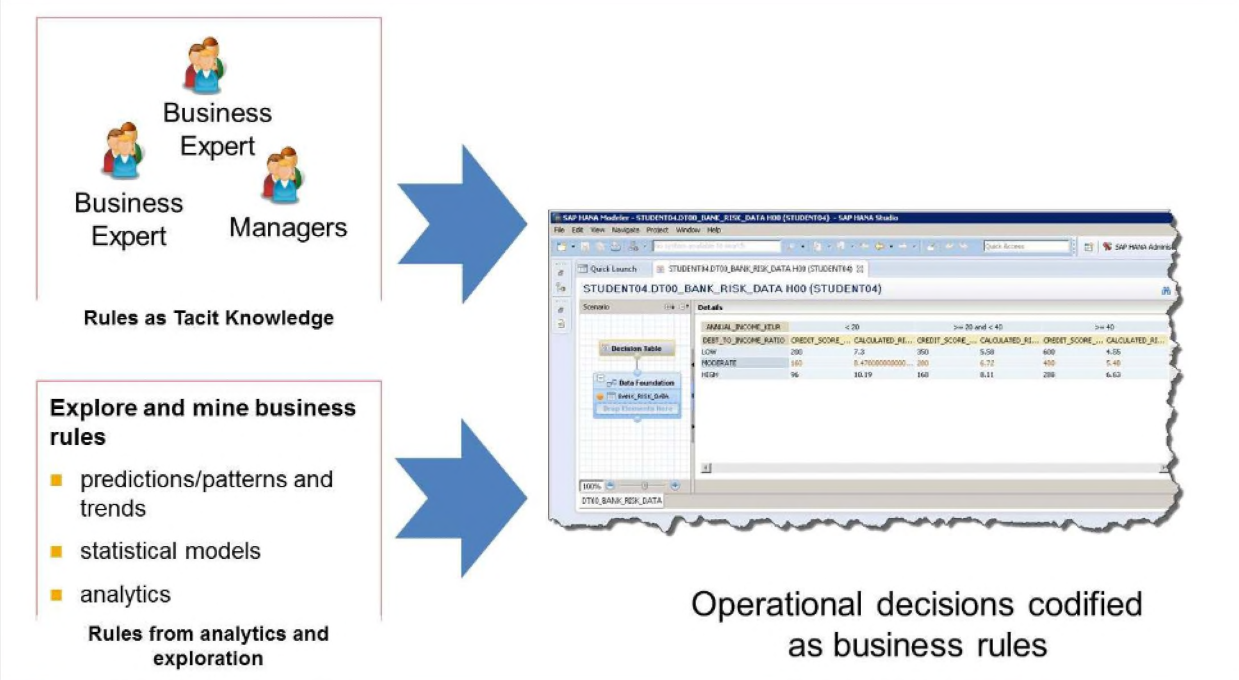


Figure 158: Different Users

Creation Workflow

The main steps to create and consume a Decision Table are as follows:

- Create a new Decision Table
- Edit the Data Foundation
- Edit the Business Rules
- Consume the Decision Table



Figure 159: Creation Workflow

Decision Table Graphical Editor

Business Rules can be graphically presented directly in the SAP HANA studio. The graphical editor for Decision Tables creation presents a Data Foundation node and a Decision Table node, as depicted in the following figure, Decision Table graphical editor.

In the Data Foundation node, you can define the vocabulary upon which you model your Business Rules. Business Rules conditions and actions to be taken are also defined in the Data Foundation node.

In the Decision Table node you can enter or upload the conditions and actions to complete the modeling of your Business Rules.

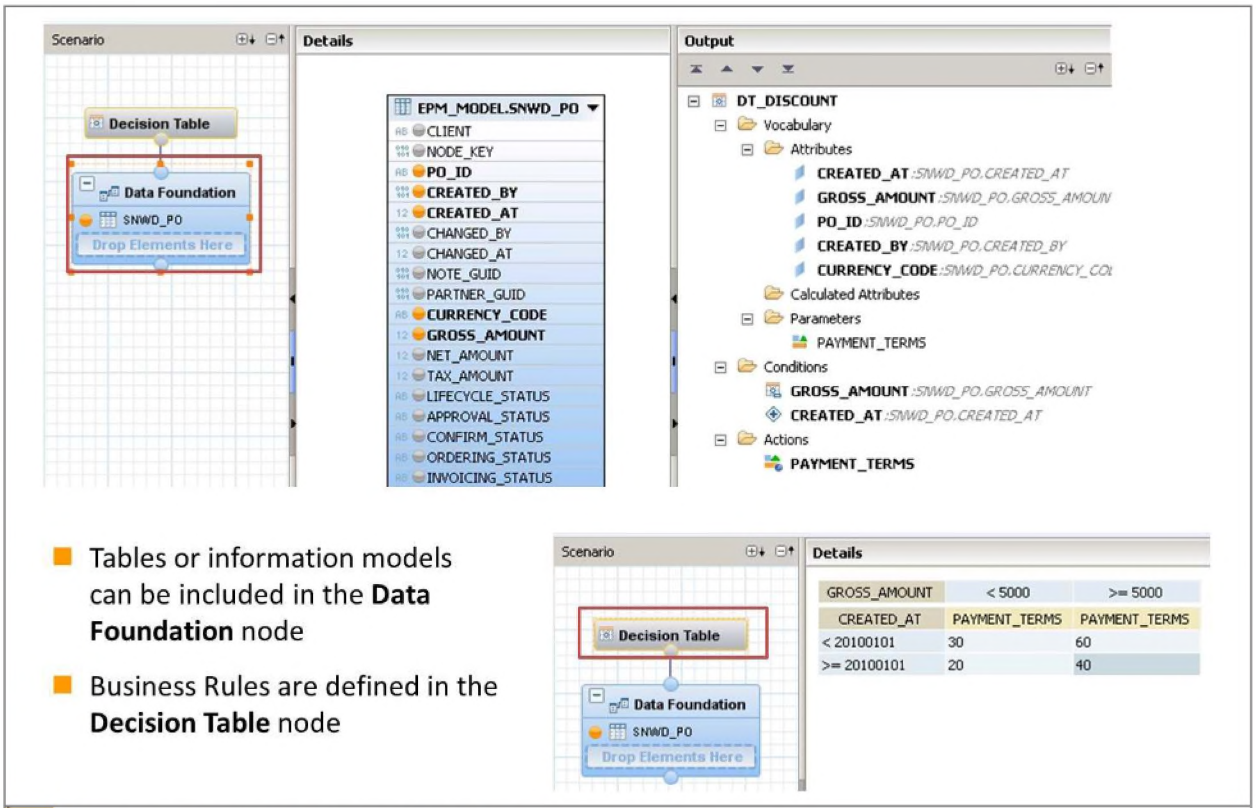


Figure 160: Decision Table Graphical Editor

Activities to be Performed in the Data Foundation Node

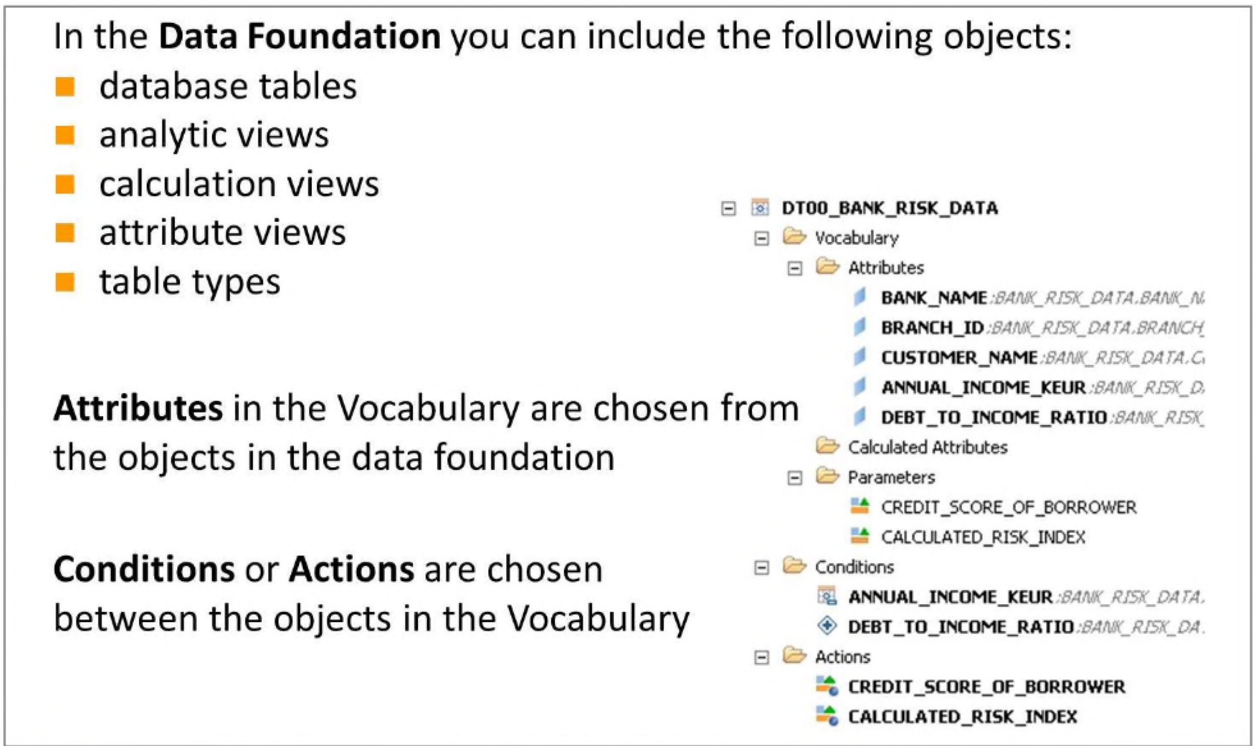
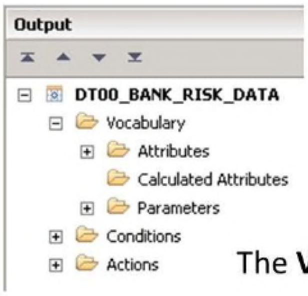


Figure 161: Activities to be Performed in the Data Foundation Node

Vocabulary, Conditions, and Actions

The building blocks of a Decision Table are the Vocabulary, Conditions, and Actions.



The **Vocabulary** contains the building blocks of a decision table.

You can include the following objects in it:

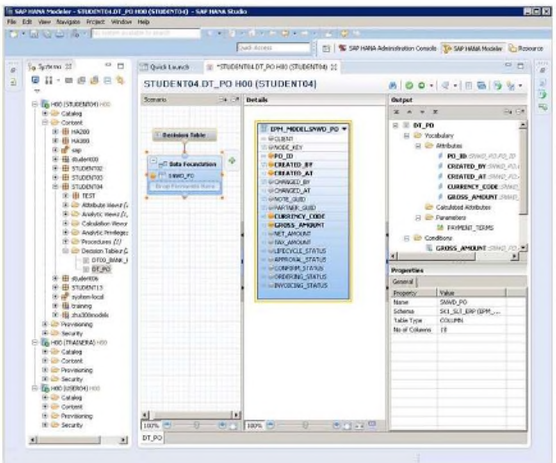
- *Attributes* (chosen from the objects in the data foundation)
- *Calculated Attributes*
- *Parameters*

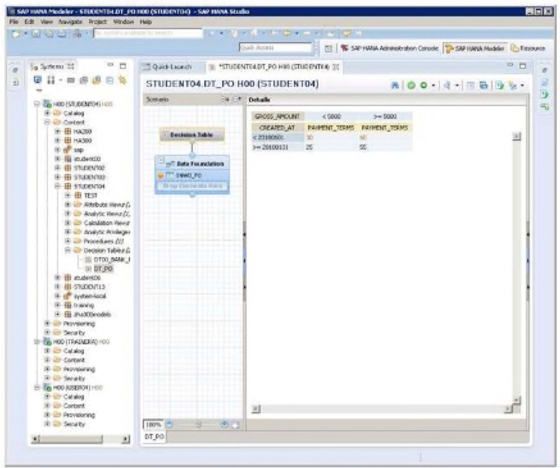
With the objects of the Vocabulary you can then define

- *Conditions* (business rules to be checked)
- *Actions* (what to do when a condition is met)
  - simulation
  - direct update

Figure 162: Vocabulary, Conditions, and Actions

Business Rules Inside SAP HANA





Business Rules Vocabulary from HANA Tables or Models

- Based on Calculation Views, Analytical Views, Attribute Views, Physical Tables and Table Types
- Ability to simulate and modify physical data based on actions (e.g. discount %)

Model – Simulate - Mine

- Possibility to model rules in a compact decision table
- Capabilities to simulate decisions and mine trends, patterns and new business rules

Figure 163: Business Rules inside SAP HANA

A Decision Table can be modeled for different purposes:

- To update a database table, based on the defined business rules

- For simulation or analytic purposes

The purpose of the Decision Table must be decided at design time.

When a Decision Table is activated, it generates a SQLScript procedure as a runtime artifact. The generated procedure can be found in the *Catalog*, in the *Procedures* folder of the *_SYS_BIC* schema. The generated procedure name will follow the pattern *<package-name>/<decisiontable-name>*. The generated procedure can be called as any other SQLScript procedure, for example in the SQL editor or within a script-based calculation view.

Executing a Decision Table in SQLScript

When calling the generated procedure for a Decision Table, all the Conditions that correspond to Parameters of the Vocabulary must be passed as input parameters to the procedure call.

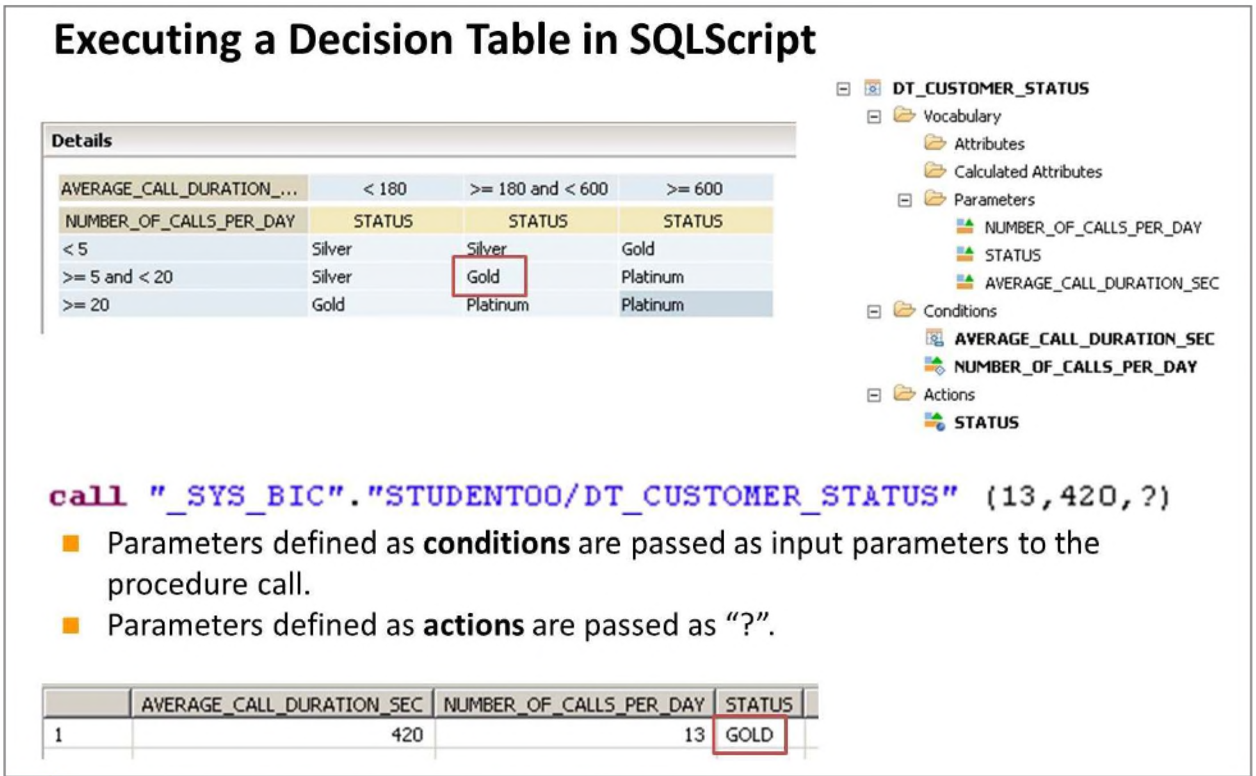


Figure 164: How to Execute a Decision Table in SQLScript

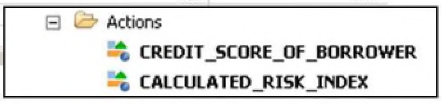
If a Decision Table is modeled on a database table for simulation purposes, all Actions can be created out of Parameters created in the Vocabulary. In this case, there is no need to pass input parameters to the generated procedure.

Executing a Decision Table on Real Data

- Executing a decision table on real data (e.g. one or more tables joined in the Data Foundation).
- Real time analysis of huge amounts of data.

```
call "_SYS_BIC"."STUDENT00/DT00_BANK_RISK_DATA"(?)
```

ANK_NAME	BRANCH_ID	CUSTOMER_NAME	ANNUAL_INCOME_KEUR	DEBT_TO_INCOME_RATIO	CREDIT_SCORE_OF_BORROWER	CALCULATED_RISK_INDEX
ank of Sweden	1	Cody	13	MODERATE	160	8.47
ank of Sweden	1	Tom	18	LOW	200	7.3
ank of Sweden	1	Dennis	20	HIGH	168	8.11
ank of Sweden	1	Robin	33	HIGH	168	8.11
ank of Sweden	1	Bedrich	35	LOW	350	5.58
ank of Sweden	1	Daniel	40	LOW	600	4.55
ank of Sweden	1	Martin	55	MODERATE	480	5.48
ank of Sweden	1	Pavel	60	LOW	600	4.55
ank of Sweden	1	Ryan	60	HIGH	288	6.63
ank of Sweden	1	Tommy	70	MODERATE		
ank of Sweden	1	Stephen	80	MODERATE		



- When all the **Actions** are chosen among the Parameters, no data is updated in the database.

Figure 165: Executing a Decision Table on Real Data

Decision Table Modeled on a Table Type

Other SAP HANA applications can consume a Decision Table as an SQLScript.

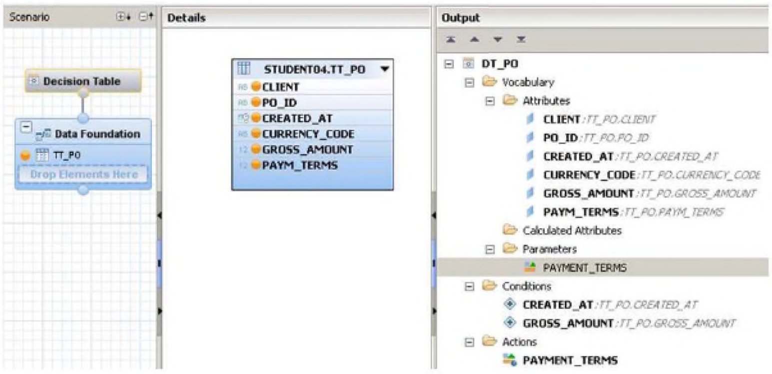
A Decision Table can also be modeled on a Table Type. This approach can be used when you want to reuse a Decision Table on different tables, in different sources, sharing the same structure. When you execute the Decision Table, you can then pass a table name as an input parameter.

A Decision Table can be modeled on a **Table Type**

- Create a Table Type

```
CREATE TYPE "STUDENT00"."TT_PO" AS TABLE ("CLIENT" NVARCHAR(3),
    "PO_ID" NVARCHAR(10) DEFAULT '0000000000',
    "CREATED_AT" DATE,
    "CURRENCY_CODE" NVARCHAR(5) NOT NULL,
    "GROSS_AMOUNT" DECIMAL(15, 2) NOT NULL);
```

- Include the Table Type in the Data Foundation



- Execute the Decision Table by passing a table name as input parameter

```
CALL "_SYS_BIC"."STUDENT00/DT_PO"('PURCHASING.PO_TABLE',?);
```

Figure 166: Decision Table Modeled on a Table Type

Creation Workflow – Create a Decision Table in a Package

To summarize the creation process of a Decision Table you need to:

- Create a New Decision Table inside a Package.
- Define the Vocabulary in the Data Foundation node, by optionally including database or modeling objects in it. Define Conditions and Actions.
- Define your Business Rules in the Decision Table node by typing or uploading the rules.
- Consume the Decision Table, for example, directly or inside a calculation view from the SAP HANA studio or from a Business Intelligence reporting tool.

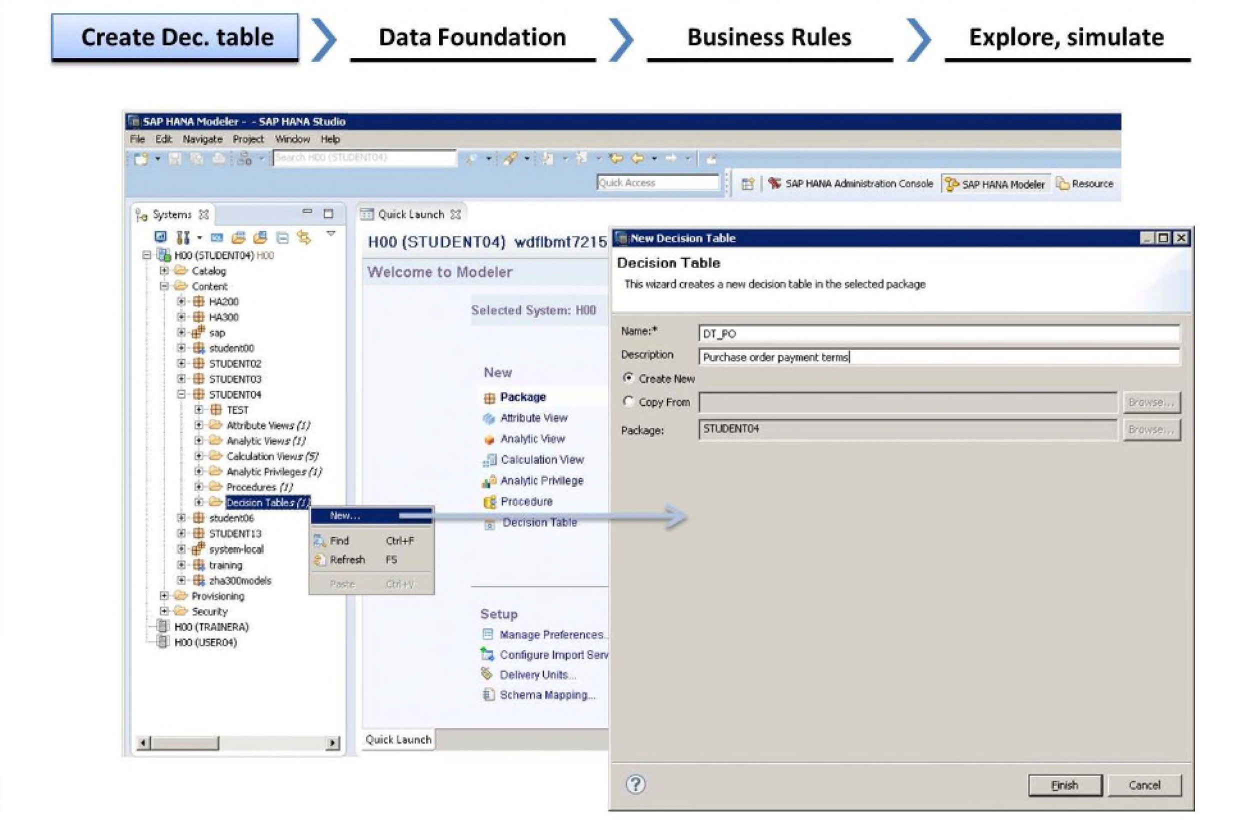
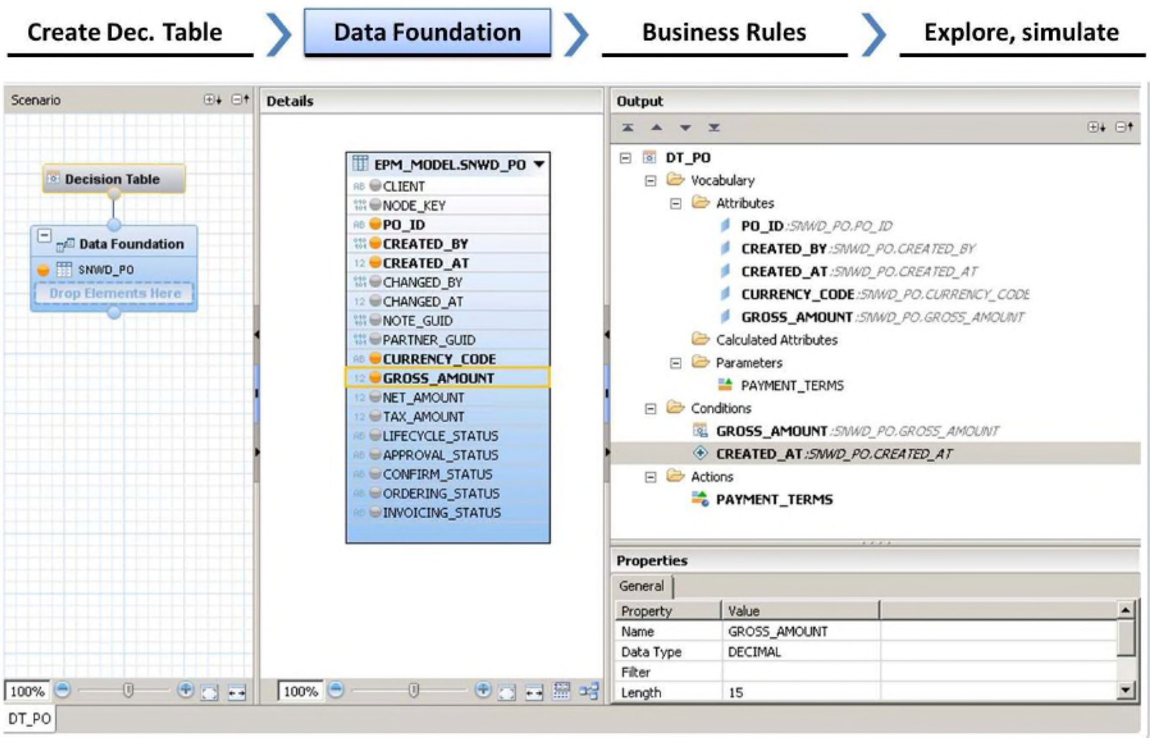


Figure 167: Creation Workflow - Create a Decision Table in a Package

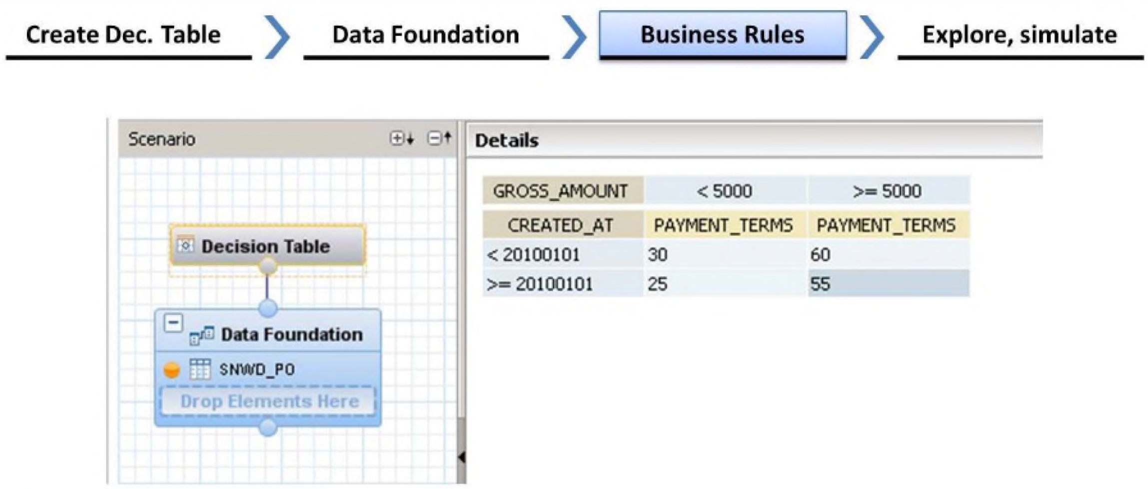
Creation Workflow – Define the Data Foundation



Choose the objects for the Data Foundation, Vocabulary, Conditions and Actions

Figure 168: Creation Workflow - Define the Data Foundation

Creation Workflow – Define Business Rules

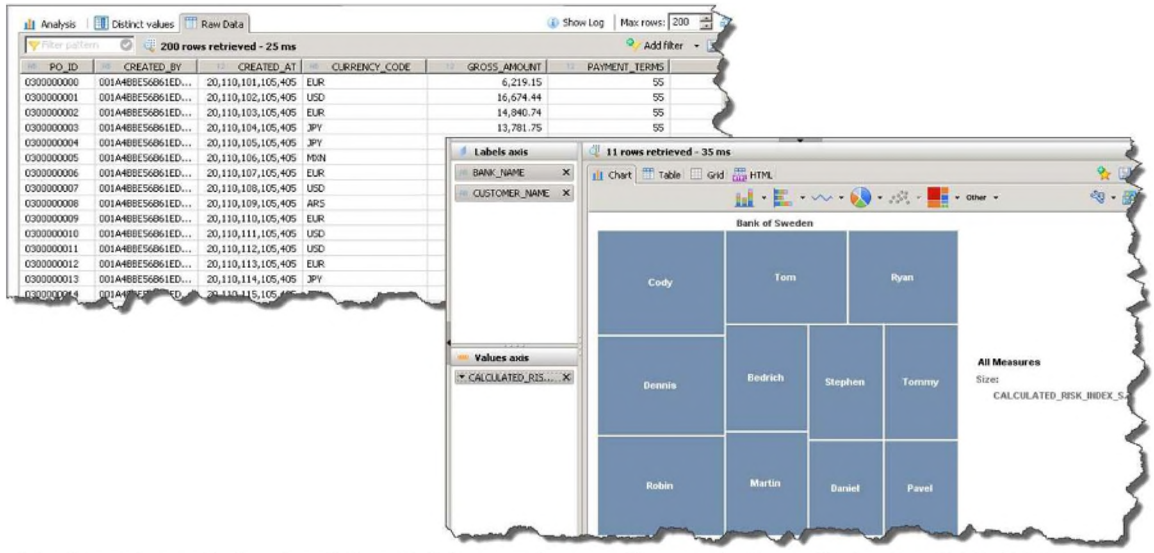


- In the **Decision Table node** of the editor, you can define conditions and actions.
- Values can be typed in manually or uploaded from a MS Excel file.
- The following operations and expressions are supported:
  - Conditional expressions
  - Date operations
  - Mathematical operations
  - Enumerations

Figure 169: Creation Workflow - Define Business Rules

Creation Workflow – Explore, Simulate, and Adapt

Create Dec. Table ❯ Data Foundation ❯ Business Rules ❯ **Explore, simulate**



Exploration of the Decision Table can be performed **directly** in the SAP HANA Studio.

The Decision Table can also be included in a **Calculation View** and thus be consumed with a Business Intelligence tool (e.g. SAP Lumira).

Figure 170: Creation Workflow - Explore, Simulate, and Adapt

From SPS10 onwards, a Decision Table can be consumed in a Calculation View by referring to the design-time object.

Before SPS10, only the runtime view (the column view in schema *_SYS_BIC* could be referenced by a node of the calculation view, thus causing an error when the Calculation View was activated before the Decision Table it consumed.

Unit 2
Exercise 14

Create a Decision Table

Exercise Objectives

After completing this exercise, you will be able to:

- Creating a Decision Table
- Define Vocabulary
- Define Business Rules
- Consume a Decision Table in a Calculation View

Business Example

You are working at a bank. The bank needs to generate credit scores and calculate a risk index for borrowers.

The Credit Scoring team has provided you with an MS Excel file in which scoring rules are represented in a matrix.

The risk measures are based on business rules that can change on short notice due to financial market movements. So, to help the bank react swiftly to market changes, you have decided to implement a Calculation View based on a Decision Table.

Task 1: Create a New Decision Table and Define its Data Foundation

1. Have a look at the source data on which you will base your business scenario. This data is located in the table *BANK_RISK_DATA* within the schema *TRAINING*.

   This table is a list of customers that mentions the Annual Income and the Debt to Income ratio.

   In particular, notice the values assigned to the *DEBT_TO_INCOME_RATIO* column: LOW, HIGH and MODERATE.

   The purpose of the Decision Table will be to automate Business decisions based on these two parameters.

2. Review the MS Excel file *HA300 → BANK_RISK_DATA_BUSINESS_RULES.xlsx* provided by the Credit Scoring team. The file is located in your *Student* folder.

   You will import this file into your model using SAP HANA studio later one.

3. Create a new Decision Table *DT_BANK_RISK_DATA_##* in your package *STUDENT##*, with the Description *Bank Risk Data*.

4. Add the table BANK_RISK_DATA from the schema TRAINING to the Data Foundation.

5. Add all the columns of the table *BANK_RISK_DATA* to the Vocabulary, as Attributes.

6. You want to calculate a credit risk and a risk index out of the data in the table *TRAINING.BANK_RISK_DATA*. The computed values should not be written in dedicated fields in the table, so you need to create two Parameters in the model for this purpose:

Parameter Name	Data Type	Type
CREDIT_SCORE_OF_BORROWER	*INTEGER*	[Leave blank]
CALCULATED_RISK_INDEX	*DECIMAL (2,2)*	[Leave blank]

> Hint:
> When creating the new Parameters, if the *OK* button remains grayed out, you can temporarily change the *Data Type* and set it again to the specified value.

7. The *annual income* and the *debt to income ratio* are the basis for the computation of the Credit score and Risk index.

   Therefore, add the following Attributes as Conditions:

   - *ANNUAL_INCOME_KEUR*
   - *DEBT_TO_INCOME_RATIO*

   Modify the *ANNUAL_INCOME_KEUR* Condition so that it will be visualized horizontally (one column for each entered condition on the annual income) in the Decision Table.

8. Set the created Parameters so that they contain the results of the execution of the Decision Table.

Task 2: Define the Business Rules of the Decision Table

The business rules create a matrix that defines the value ranges (or discrete values) for each condition, and assigns values (one value for each Action) to each pair (or n-tuple) of conditions.

Here the purpose is to define business rules, for example, what will be the credit score and credit risk index of a borrower if the annual income is more than 40 KEUR and the debt to income ratio moderate.

1. Enter the three following Condition Values for the *ANNUAL_INCOME_KEUR* condition:

   - `< 20`
   - `>= 20 AND < 40`
   - `>= 40`

> Hint:
> You must right-click the Condition name and choose *Add Condition Values*.

2. Similarly, enter the following Values for the *DEBT_TO_INCOME_RATIO* condition:

   - **LOW**
   - **MODERATE**

- **HIGH**

In this case, you can use the list of values that are proposed (based on the corresponding column of the Data foundation).

> Hint:
> In the *Add Condition Values* screen, select the first row and click the ellipsis *(...)* button.

3. Delete the default Condition Values for both Conditions:
   - *ANNUAL_INCOME_KEUR*: *0*
   - *DEBT_TO_INCOME_RATIO Default Value*

Task 3: Import the Business Rules from an MS Excel File

1. Make sure your Decision Table tab looks as follows: *(the figure, Decision Table, is partly showing the results).*

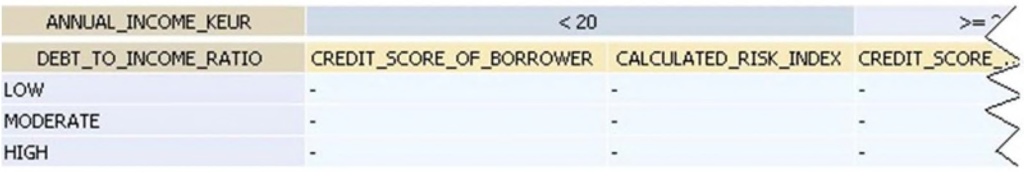


Figure 173: Decision Table

> Note:
> To avoid issues when importing the business rules, pay attention to the spelling of the parameters that you have created.

2. Instead of manually typing the Business Rules, upload them from the MS Excel file *HA300 → BANK_RISK_DATA_BUSINESS_RULES.xlsx*.

> Hint:
> Right-click anywhere in the *Details* pane.

   Your Decision Table is populated with business rules.

3. Save and Activate your Decision Table.

Task 4: Consume the Data

1. You want to include your Decision Table into a Calculation View, in order to expose it to different Business Intelligence reporting tools, such as SAP Lumira.

   To do so, create a new graphical Calculation View **CV_BANK_RISK_DATA_##**, based on the Decision Table *DT_BANK_RISK_DATA_##* that you have created in the previous steps.

> Note:
> Up to release SPS09, it was not possible to add a design-time Decision Table directly to a Calculation View. Instead, you had to use the corresponding Column View, generated in the *_SYS_BIC* schema when activating the Decision Table. This column view is: *_SYS_BIC.STUDENT##/DT_BANK_RISK_DATA_##/RV*, and the following steps to define output columns, semantics, and so on, would be similar.

2. In the Aggregation node, add all columns to output.

> Caution:
> Do NOT add the columns to the output as *Aggregated Columns*.

3. Assign the Type *Attribute* to the following columns:
   - BANK_NAME
   - BRANCH_ID
   - CUSTOMER_NAME
   - DEBT_TO_INCOME_RATIO
4. Assign the Type *Measure* to the following columns:
   - ANNUAL_INCOME_KEUR
   - CREDIT_SCORE_OF_BORROWER
   - CALCULATED_RISK_INDEX
5. Save and Activate the Calculation View.
6. Preview the data and check that it is similar to the data preview of the Decision Table.
   The computation of your decision table is now ready for consumption with any BI client.

Create a Decision Table

Exercise Objectives

After completing this exercise, you will be able to:

- Creating a Decision Table
- Define Vocabulary
- Define Business Rules
- Consume a Decision Table in a Calculation View

Business Example

You are working at a bank. The bank needs to generate credit scores and calculate a risk index for borrowers.

The Credit Scoring team has provided you with an MS Excel file in which scoring rules are represented in a matrix.

The risk measures are based on business rules that can change on short notice due to financial market movements. So, to help the bank react swiftly to market changes, you have decided to implement a Calculation View based on a Decision Table.

Task 1: Create a New Decision Table and Define its Data Foundation

1. Have a look at the source data on which you will base your business scenario. This data is located in the table *BANK_RISK_DATA* within the schema *TRAINING*.

   This table is a list of customers that mentions the Annual Income and the Debt to Income ratio.

   In particular, notice the values assigned to the *DEBT_TO_INCOME_RATIO* column: LOW, HIGH and MODERATE.

   The purpose of the Decision Table will be to automate Business decisions based on these two parameters.

   a) In the Systems view, open the *Catalog* → *TRAINING* → *Tables* folder.

   b) Right-click the *BANK_RISK_DATA* table and choose *Open Content*.

2. Review the MS Excel file *HA300* → *BANK_RISK_DATA_BUSINESS_RULES.xlsx* provided by the Credit Scoring team. The file is located in your *Student* folder.

   You will import this file into your model using SAP HANA studio later one.

   a) From your *Student* folder, double click the file *HA300* → *BANK_RISK_DATA_BUSINESS_RULES.xlsx*.

   b) Review the structure of the Excel file.

3. Create a new Decision Table *DT_BANK_RISK_DATA_##* in your package *STUDENT##*, with the Description *Bank Risk Data*.
   a) In the *SAP HANA Modeler* perspective, right-click your package *STUDENT##* and choose *New → Decision Table*.
   b) Enter the Name and Description of the Decision Table.
   c) Choose *Finish*.
4. Add the table BANK_RISK_DATA from the schema TRAINING to the Data Foundation.
   a) In the Scenario pane, right-click the Data Foundation node and choose *Add Objects*.
   b) In the *Find* dialog box, enter **BANK**.
   c) Select the BANK_RISK_DATA (TRAINING) table and choose OK.
5. Add all the columns of the table *BANK_RISK_DATA* to the Vocabulary, as Attributes.
   a) In the *Details* pane, right-click the name of the *TRAINING.BANK_RISK_DATA* object and choose *Add All as Attribute*.
6. You want to calculate a credit risk and a risk index out of the data in the table *TRAINING.BANK_RISK_DATA*. The computed values should not be written in dedicated fields in the table, so you need to create two Parameters in the model for this purpose:

Parameter Name	Data Type	Type
CREDIT_SCORE_OF_BOR-ROWER	*INTEGER*	[Leave blank]
CALCULATED_RISK_INDEX	*DECIMAL (2,2)*	[Leave blank]

> Hint:
> When creating the new Parameters, if the *OK* button remains grayed out, you can temporarily change the *Data Type* and set it again to the specified value.

   a) In the *Output* pane, right-click the *Parameters* folder and choose *New*. Enter the details from the table to create the first parameter.
   b) Choose *OK*.
   c) Repeat the previous steps for the other parameter.
7. The *annual income* and the *debt to income ratio* are the basis for the computation of the Credit score and Risk index.

   Therefore, add the following Attributes as Conditions:

   - *ANNUAL_INCOME_KEUR*
   - *DEBT_TO_INCOME_RATIO*

   Modify the *ANNUAL_INCOME_KEUR* Condition so that it will be visualized horizontally (one column for each entered condition on the annual income) in the Decision Table.

   a) Right-click each of the Attributes *ANNUAL_INCOME_KEUR* and *DEBT_TO_INCOME_RATIO* and choose *Add as Conditions*.

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

b) In the Conditions folder, right-click *ANNUAL_INCOME_KEUR* and choose *Mark as Horizontal Condition*.

> Note:
> You will see later on how you can modify the layout of the Decision Table.

8. Set the created Parameters so that they contain the results of the execution of the Decision Table.

   a) Right-click each of the parameters from the *Parameter* folder and choose *Add as Actions*.

Task 2: Define the Business Rules of the Decision Table

The business rules create a matrix that defines the value ranges (or discrete values) for each condition, and assigns values (one value for each Action) to each pair (or n-tuple) of conditions.

Here the purpose is to define business rules, for example, what will be the credit score and credit risk index of a borrower if the annual income is more than 40 KEUR and the debt to income ratio moderate.

1. Enter the three following Condition Values for the *ANNUAL_INCOME_KEUR* condition:

   - `< 20`
   - `>= 20 AND < 40`
   - `>= 40`

> Hint:
> You must right-click the Condition name and choose *Add Condition Values*.

   a) In the *Scenario* pane, choose the *Decision Table* node.

   b) Right-click the *ANNUAL_INCOME_KEUR* cell the right (not below) and choose *Add Condition Values*. Enter the business rules as specified.

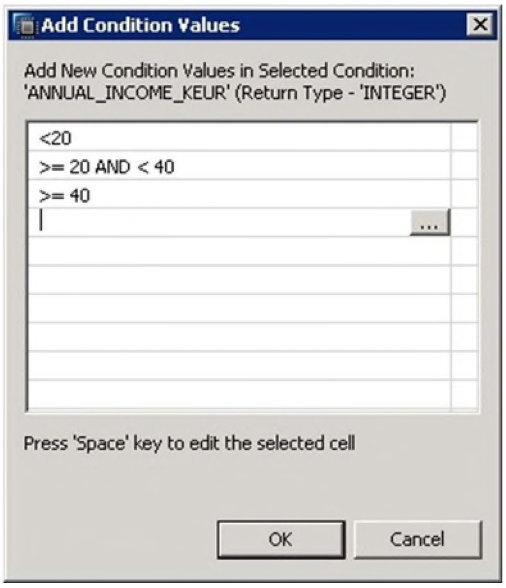


Figure 171: Condition Values for ANNUAL_INCOME_KEUR

c) To delete the zero value cell, right-click and select *Delete Condition Value* from the pop-up menu.

2. Similarly, enter the following Values for the *DEBT_TO_INCOME_RATIO* condition:

- **LOW**
- **MODERATE**
- **HIGH**

In this case, you can use the list of values that are proposed (based on the corresponding column of the Data foundation).

> Hint:
> In the *Add Condition Values* screen, select the first row and click the ellipsis *(...)* button.

a) Right-click below *DEBT_TO_INCOME_RATIO* in the Decision Table tab and select *Add Condition Values*. Enter the business rules as follows:

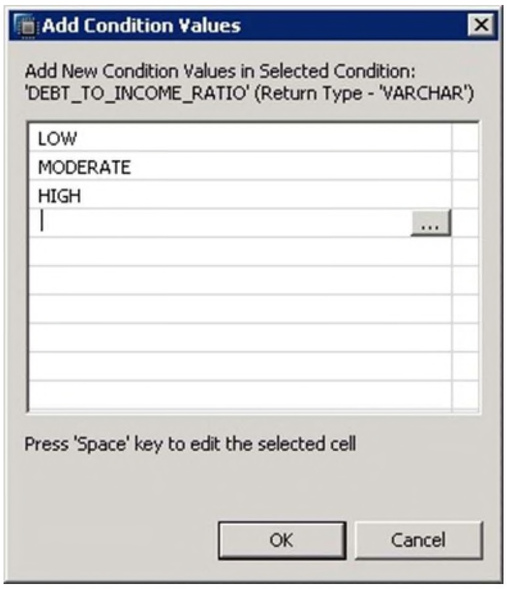


Figure 172: Condition Values for DEBT_TO_INCOME_RATIO

3. Delete the default Condition Values for both Conditions:
   - *ANNUAL_INCOME_KEUR*: *0*
   - *DEBT_TO_INCOME_RATIO Default Value*

   a) Right-click the default condition values, and choose *Delete* and confirm the deletion.

Task 3: Import the Business Rules from an MS Excel File

1. Make sure your Decision Table tab looks as follows: *(the figure, Decision Table, is partly showing the results)*.

ANNUAL_INCOME_KEUR	< 20		>= ...
DEBT_TO_INCOME_RATIO	CREDIT_SCORE_OF_BORROWER	CALCULATED_RISK_INDEX	CREDIT_SCORE_...
LOW	-	-	-
MODERATE	-	-	-
HIGH	-	-	-

Figure 173: Decision Table

Note:
To avoid issues when importing the business rules, pay attention to the spelling of the parameters that you have created.

2. Instead of manually typing the Business Rules, upload them from the MS Excel file *HA300 → BANK_RISK_DATA_BUSINESS_RULES.xlsx*.

Hint:
Right-click anywhere in the *Details* pane.

Your Decision Table is populated with business rules.

   a) Right-click anywhere in the *Details* pane and choose *Import Data from MS Excel*.

b) Locate the file *HA300 → BANK_RISK_DATA_BUSINESS_RULES.xlsx*, select it and click *Open*.

3. Save and Activate your Decision Table.

   a) Choose *Save and Activate*.

Task 4: Consume the Data

1. You want to include your Decision Table into a Calculation View, in order to expose it to different Business Intelligence reporting tools, such as SAP Lumira.

   To do so, create a new graphical Calculation View **CV_BANK_RISK_DATA_##**, based on the Decision Table *DT_BANK_RISK_DATA_##* that you have created in the previous steps.

   > Note:
   > Up to release SPS09, it was not possible to add a design-time Decision Table directly to a Calculation View. Instead, you had to use the corresponding Column View, generated in the *_SYS_BIC* schema when activating the Decision Table. This column view is: *_SYS_BIC.STUDENT##/DT_BANK_RISK_DATA_##/RV*, and the following steps to define output columns, semantics, and so on, would be similar.

   a) In the *Systems* view, right-click your package *STUDENT##* and choose *New → Calculation View*.

   b) Enter the name of the Calculation View and leave the default values for all other fields.

   c) Choose *Finish*

   d) Add the Decision Table *DT_BANK_RISK_DATA_##* from your package *STUDENT##* to the Aggregation node.

   > Note:
   > With SAP HANA SPS09 or earlier, you would add the column view *_SYS_BIC.STUDENTxx/DT_BANK_RISK_DATA_##/RV* to the Aggregation node.

2. In the Aggregation node, add all columns to output.

   > Caution:
   > Do NOT add the columns to the output as *Aggregated Columns*.

   a) Select the *Aggregation* node.

   b) In the *Details* pane, select all the columns from the Decision Table, right-click, and choose *Add To Output*.

   > Hint:
   > You can also right-click the header of the column list and choose *Add All To Output*.

3. Assign the Type *Attribute* to the following columns:

   - BANK_NAME
   - BRANCH_ID
   - CUSTOMER_NAME
   - DEBT_TO_INCOME_RATIO

   a) In the *Scenario* pane, select the *Semantics* node.

   b) In the Details pane, double-click the Type of the *BANK_NAME* column and choose *Attribute*.

   c) Repeat this step for the other columns.

   > Hint:
   > You can also select all the 4 columns while holding the **`Ctrl`** key, and click the *Mark as Attribute* button on the top-right corner of the *Columns* tab.

4. Assign the Type *Measure* to the following columns:

   - ANNUAL_INCOME_KEUR
   - CREDIT_SCORE_OF_BORROWER
   - CALCULATED_RISK_INDEX

   a) In the *Scenario* pane, select the *Semantics* node.

   b) In the Details pane, double-click the Type of the ANNUAL_INCOME_KEUR column and choose *Measure*.

   c) Repeat this step for the other columns.

5. Save and Activate the Calculation View.

   a) Choose *Save and Activate*.

6. Preview the data and check that it is similar to the data preview of the Decision Table.

   a) Choose *Data Preview*.

   The computation of your decision table is now ready for consumption with any BI client.

LESSON SUMMARY

You should now be able to:

- Create decision tables
- Consume decision tables in Calculation Views

Learning Assessment

1. What are the features of parent-child hierarchy?

   Choose the correct answers.

   - [ ] A A single dedicated column is used to store the parent of each record of the table.
   - [ ] B Heterogeneous fields are combined in a hierarchy.
   - [ ] C Parent and child field have the same data type.
   - [ ] D Recursive data structure defines the hierarchy.

2. Restricted Columns provide a subset of the original column.

   Determine whether this statement is true or false.

   - [ ] True
   - [ ] False

3. Which of the following benefits characterize the use of a design time Filter?

   Choose the correct answers.

   - [ ] A It reduces the result set of data.
   - [ ] B The filter is applied on the results set of a query.
   - [ ] C It is defined on the runtime in the SQL query.
   - [ ] D The filter is applied on the table before the query is executed.
   - [ ] E It is applied before a table join is executed.

4. When the *Default Client* property of an information view is set to *Session Client*, the data is filtered dynamically based on the CLIENT assigned to the user.

   Determine whether this statement is true or false.

   - [ ] True
   - [ ] False

5. Identify the object that adds a WHERE clause to the query.

   Choose the correct answer.

   - [ ] A Modeler
   - [ ] B Attributes
   - [ ] C Input parameter
   - [ ] D Variable

6. In a Calculation View, for conversion purposes, you can use input parameters as placeholders and formulas like calculated columns.

   Determine whether this statement is true or false.

   - [ ] True
   - [ ] False

7. While converting currencies, which parameter is as important as source and target currencies?

   Choose the correct answer.

   - [ ] A Money
   - [ ] B Time
   - [ ] C Location
   - [ ] D Season

8. What are the currency conversion functionalities supported by SAP HANA?

   Choose the correct answers.

   - [ ] A Conversion based on customer defined tables
   - [ ] B Enable for decimal shifts
   - [ ] C Determination of target currency based on attributes or input parameters
   - [ ] D Multiple Exchange Rate Types

9. Which of the following processes is involved in the creation of a Decision Table?

Choose the correct answer.

- [ ] A Define Vocabulary > Define Business Rules > Create a Decision Table > Explore, Simulate, and Adapt
- [ ] B Define Business Rules > Define Vocabulary > Explore, Simulate, and Adapt > Create a Decision Table
- [ ] C Create a Decision Table > Define Vocabulary > Define Business Rules > Explore, Simulate, and Adapt
- [ ] D Explore, Simulate, and Adapt> Define Business Rules and Vocabulary> Create a Decision Table

Unit 2

Learning Assessment - Answers

1. What are the features of parent-child hierarchy?

   Choose the correct answers.

   - [x] A A single dedicated column is used to store the parent of each record of the table.
   - [ ] B Heterogeneous fields are combined in a hierarchy.
   - [x] C Parent and child field have the same data type.
   - [x] D Recursive data structure defines the hierarchy.

2. Restricted Columns provide a subset of the original column.

   Determine whether this statement is true or false.

   - [x] True
   - [ ] False

3. Which of the following benefits characterize the use of a design time Filter?

   Choose the correct answers.

   - [x] A It reduces the result set of data.
   - [ ] B The filter is applied on the results set of a query.
   - [ ] C It is defined on the runtime in the SQL query.
   - [x] D The filter is applied on the table before the query is executed.
   - [x] E It is applied before a table join is executed.

4. When the *Default Client* property of an information view is set to *Session Client*, the data is filtered dynamically based on the CLIENT assigned to the user.

   Determine whether this statement is true or false.

   - [x] True
   - [ ] False

5. Identify the object that adds a WHERE clause to the query.

   Choose the correct answer.

   [ ] A Modeler

   [ ] B Attributes

   [ ] C Input parameter

   [X] D Variable

6. In a Calculation View, for conversion purposes, you can use input parameters as placeholders and formulas like calculated columns.

   Determine whether this statement is true or false.

   [X] True

   [ ] False

7. While converting currencies, which parameter is as important as source and target currencies?

   Choose the correct answer.

   [ ] A Money

   [X] B Time

   [ ] C Location

   [ ] D Season

8. What are the currency conversion functionalities supported by SAP HANA?

   Choose the correct answers.

   [ ] A Conversion based on customer defined tables

   [X] B Enable for decimal shifts

   [X] C Determination of target currency based on attributes or input parameters

   [X] D Multiple Exchange Rate Types

9. Which of the following processes is involved in the creation of a Decision Table?

   Choose the correct answer.

   - [ ] A Define Vocabulary > Define Business Rules > Create a Decision Table > Explore, Simulate, and Adapt
   - [ ] B Define Business Rules > Define Vocabulary > Explore, Simulate, and Adapt > Create a Decision Table
   - [x] C Create a Decision Table > Define Vocabulary > Define Business Rules > Explore, Simulate, and Adapt
   - [ ] D Explore, Simulate, and Adapt> Define Business Rules and Vocabulary> Create a Decision Table

UNIT 3 SQL Script and Procedures

UNIT OBJECTIVES

- Describe SAP HANA SQL
- Work with SQLScript
- Explain the SQLScript implementation logic
- Create and use procedures
- Create a procedure
- Call a procedure
- Use SQLScript Debug Tools
- Understand Deprecated Modeling Objects

Unit 3
Lesson 1

Introducing SAP HANA SQL

LESSON OVERVIEW

This lesson provides an introduction to the SQL Language in SAP HANA, and covers the following topics:

- Overview of SQL Language Elements
- Identifiers
- SQL Data Types
- Predicates and Operators
- Functions and Expressions
- SQL Statement Examples

Business Example

As a consultant you want to perform tasks such as querying source data out of a table or an information model, granting or revoking privileges, copying tables, or altering table column types. Knowledge of SQL will let you perform these tasks.

SAP HANA studio is essentially an SQL generator from which commands can be sent to the SAP HANA database graphically using the user interface, but also directly using the SQL editor in the studio or by creating functions and procedures.

Previously in SAP HANA, you had to write a lot more SQL or SQLScript. Once you started using SQL or SQLScript, there was no way to take this back to the graphical models. You had to complete the modeling by writing code. As from SPS9 and SPS10, you can call the code you wrote from inside the graphical modeling environment. The result is that you now only write the code that you require in SQL or SQLScript, and then carry on again with graphical modeling. This makes modeling much easier and a lot faster to create.

> Note:
> Aim to write a minimum amount of SQL and SQLScript code, and to re-use the code you write in your graphical calculation models.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe SAP HANA SQL

SAP HANA SQL - Definition and Terminology

SQL Structured Query Language

- Standardized language for communication with a relational database.
- Used to retrieve, store or manipulate information in the database.

Class	Description	Example
DDL	Data Definition Language	CREATE, ALTER, DROP TABLE
DML	Data Manipulation Language	SELECT, DELETE, INSERT, UPDATE
DCL	Data Control Language	GRANT, REVOKE

Figure 174: SQL - Structured Query Language

SQL stands for Structured Query Language.

> Note:
> You often hear people pronounce SQL as "SEQUEL", but this is not correct. The language actually used to be called SEQUEL many years ago and so that pronunciation was correct at that time. But since the language was changed to SQL, you simply say the letters.

SQL is a hugely popular and standardized language, defined by the American National Standards Institute (ANSI), to define a database, read, update and insert data, and define security to relational databases.

SQL statements (or commands) are grouped into three classifications which are:

- DDL — Data definition language is a group of SQL statements to create, modify, and delete database objects such as tables, views, and procedures.
- DML — Data manipulation language is a group of SQL statements to insert, update, delete, and read the data records.
- DCL — Data control language is a group of SQL statements to define the database users, roles, and to grant security clearance to the database objects. (For example, who can access a particular table?)

SQL is a descriptive (sometimes also called declarative) language, in contrast to procedural languages. With SQL you can express what you want to achieve, rather than how to achieve it.

SQL really doesn't like to be told how to do something. So just tell SQL what you want and it figures out the most optimal way to achieve the required results. Depending on the growth of the data, the optimal way might change over time.

The SAP HANA SQL, including SQLScript, developed by SAP is an extension of the ANSI SQL. This was developed to better deal with the specific in-memory nature of the SAP HANA database. Standard ANSI SQL statements are of course supported and in addition extensions are available. For example, in the DDL component, SQLScript allows you to create row based or column based tables. As an additional example of extensions provided by SQLScript, we have flow control logic available, for example with the `IF/ELSE` construct, which is not available in ANSI SQL.

SQLScript can be used in different SAP HANA objects, such as the following:

- Stored Procedures
- Database Triggers
- Table Functions
- Dynamic Analytic Privileges
- Functions

SQL Console

The SQL Console can be used to type in SQL statements and directly execute them.

This method of working with SQL is not really recommended. There are a few problems, as follows, with this way of doing things:

- The statements are not saved anywhere. So you have to re-type the same statements later in the Test and Production systems.
- You will also need the authorizations to execute these statements in the Test and Production systems, leading to potential security issues. The security auditors will definitely not like this.

A better way is to use Core Data Services (CDS) or to create design-time artifacts like used-defined functions or procedures. You can transport these artifacts from the Development system to the Testing and Production systems, you do not have to re-do the work, and it addresses the security issues.

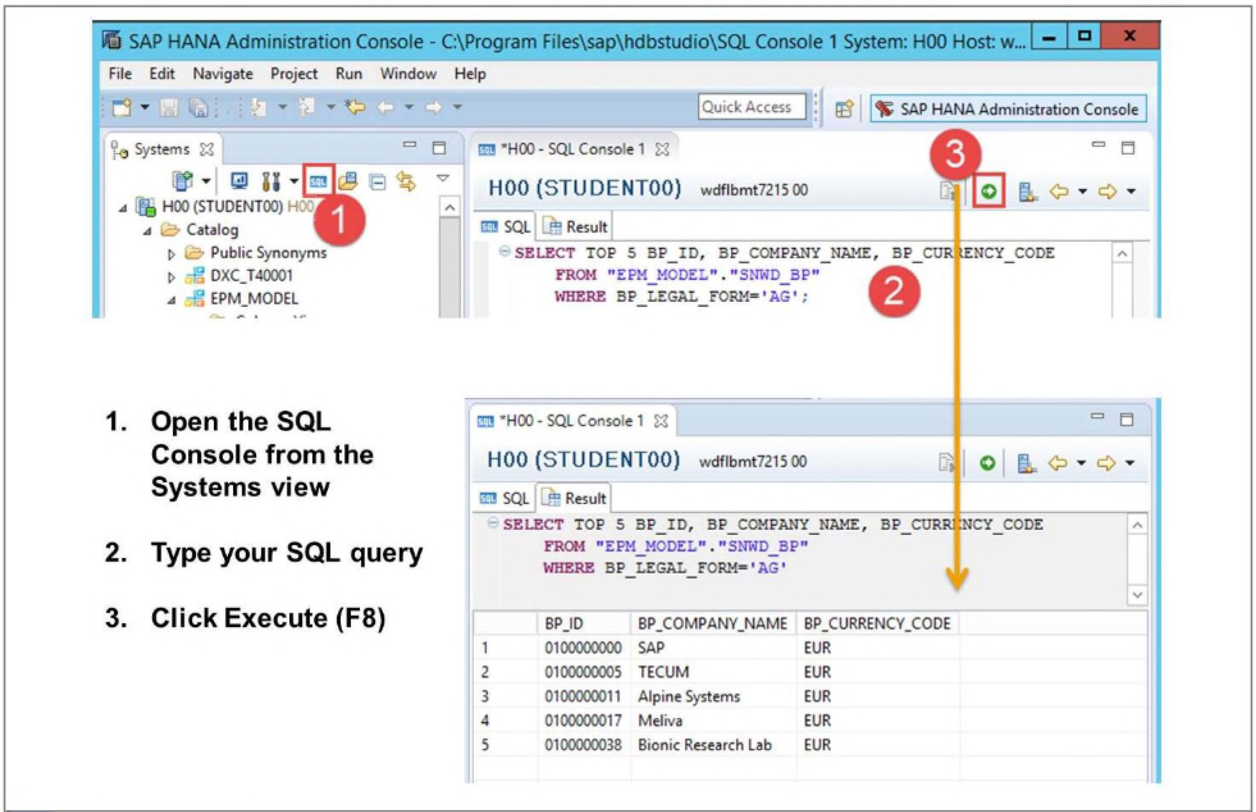


Figure 175: SQL Console

The SQL Console can be opened from the Systems view in the HANA Studio.

SQL Console Autocomplete



Figure 176: SQL Console Autocomplete

The SQL Console includes an auto-completion feature, which can be activated by pressing CTRL + SPACE while entering a command.

The auto-complete feature is also available in the code editor when you create functions and procedures.

Example of an SQL Statement

```
-- Find a list of SAP courses about HANA

SELECT name, description, cost
FROM courses
WHERE company = 'SAP'
      AND description LIKE '%HANA%';
/*

   Look at the comments, SELECT statement,
   identifiers (name, description, cost, company),
   predicates (= and LIKE), and operators (AND)

*/
```

Figure 177: Example of an SQL Statement

Let's have a look the example SQL statement in the figure, Example of an SQL Statement:

- Comments – All text after a double hyphen (`--`) or between the `/*` and `*/` text is seen as comments. In this case, you specified in the top line what the statement does.
- `SELECT` statement – The `SELECT` statement is the most used statement when writing SQL code. It allows you to read data from a data source.
  - You first specify which columns (fields) you want to read from the data source. In this case, you want to read the name of the course, the course description, and the cost of the course. You can read ALL the columns (fields) from a data source by writing `SELECT *`. This is however not recommended, especially for columnar databases like SAP HANA, as can lead to bad performance!
  - You can specify the name of the data source in the `FROM` clause, which in this case is a table named courses.
  - You can then restrict the amount of data returned by specifying a `WHERE` clause. In this case you only want to see courses from SAP.
  - You can further restrict the data returned by additional clauses using the `AND` and `OR` operators.
  - Finally you can specify that you only want SAP courses where the description field contains the word HANA somewhere in the text. You do this using the `LIKE` predicate.
- The field names in this statement can be referred to as identifiers.

SQL Language Elements

Table 10: SQL Language Elements

Element	Description
Identifiers	Used to represent names used in SQL statements.
Data type	Specify the characteristics of a data value.
Expressions	Clause that can be evaluated to return values.
Operators	Used for calculation, value comparison, or to assign values.
Predicates	Specified by combining one or more expressions or logical operators and returns one of the following logical truth values: `TRUE`, `FALSE`, or `UNKNOWN`.
Functions	Used in expressions to return information from the database.

Comment and Code Page

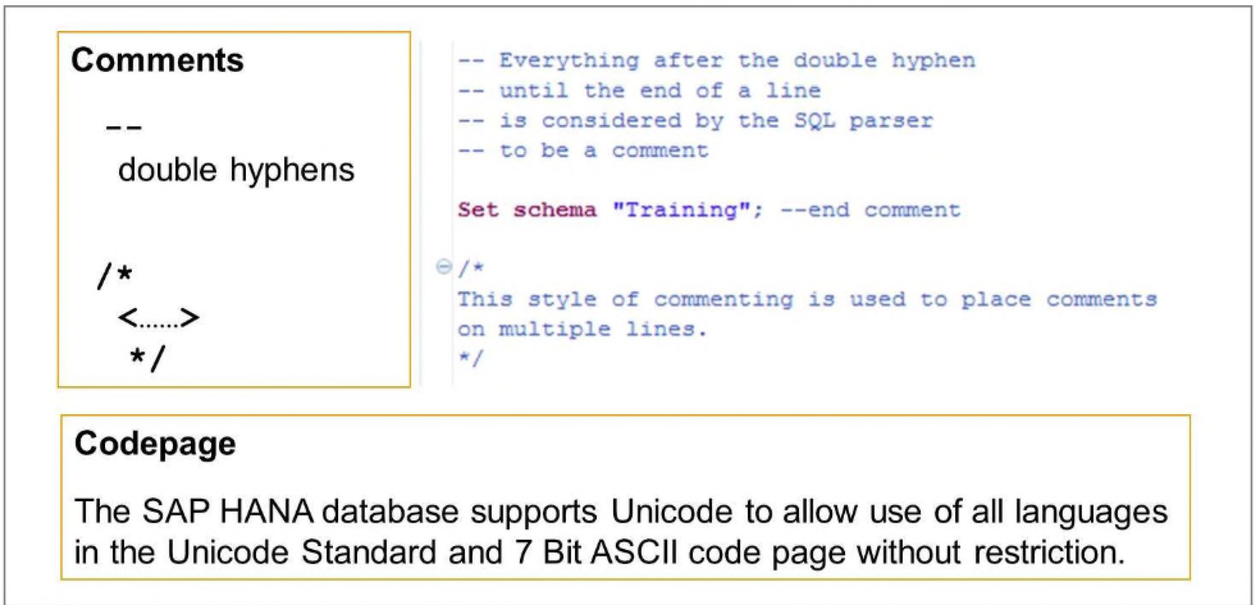


Figure 178: Comment and Code Page

Comments are delimited in SQL statements as follows:

- Double hyphens `--`. Everything after the double hyphen until the end of a line is considered by the SQL parser as a comment
- `/*` and `*/`. This style of commenting is used to place comments on multiple lines.

The SAP HANA Database supports Unicode to allow use of all languages in the Unicode Standard and 7 Bit ASCII code page without restriction.

Identifiers

Undelimited Identifiers	• must start with a letter • cannot contain any symbols other than digits or an underscore "_	TRAINING Training (treated with upper case!) ~~1TRAINING~~ ~~TRAINING%~~
Delimited Identifiers	• enclosed in the delimiter double quotes "<identifier>" • can contain any character including special characters.	"TRAINING" "Training" (case sensitive) "1Training" "Training%"
Limitations	• "_SYS_" is reserved exclusively for database engine • Role name and user name must be specified as undelimited identifiers. • Maximum length for the identifiers is 127 characters.	

Figure 179: Identifiers

Identifiers are used to represent names used in SQL statement including the following:

- table name
- view name
- synonym name
- column name
- index name
- function name
- procedure name
- user name
- role name, and so on

There are two kinds of identifiers; undelimited identifiers and delimited identifiers.

- Undelimited table and column names must start with a letter and cannot contain any symbols other than digits or an underscore _.

> Note:
> Undelimited identifiers are implicitly treated as upper case. When quoting identifiers, it is possible to use spaces and lower or mixed case in identifiers.

- Delimited identifiers are enclosed in the delimiter, double quotes, then the identifier can contain any character including special characters.

- For example, →AB$%CD→ is a valid identifier name.

> Note:
> Delimited identifiers are case sensitive! SAP HANA is case-sensitive on database objects like the names of tables and fields. For example, you can have a table called "DATA" and another called "data". These will be treated like two different tables.

- Single Quotation Mark

  Single quotation marks are used to delimit string literals and a single quotation mark itself can be represented using two single quotation marks.

- Double Quotation Mark

  Double quotation marks are used to delimit identifiers and double quotation mark itself can be represented using two double quotation marks.

Example

```
select 'String',
'SAP''s
current offering' from "DUMMY"
```

Data Types

Data types are used to specify the characteristics of the data stored in the database, for example, a string can be stored as a NVARCHAR data type.

Table 11: SQL Data Types

Classification	Standard ANSi-92 Data Type
Date/Time	DATE, TIME, TIMESTAMP
Numeric	DECIMAL, INTERGER, SMALLINT, FLOAT, REAL, DOUBLE
Character String	CHAR, NCHAR, VARCHAR, NVARCHAR
Binary	BINARY, VARBINARY

Data type specifies the characteristics of a data value.

> Note:
> A special value of NULL is included in every data type to indicate the absence of a value.

Predicates

Predicates are typically used in the `WHERE` clause of a `SELECT` statement.

Comparison Predicates	**<expression> { = | != | <> | > | < | >= | <= } [ANY | SOME | ALL] { <expression_list> | <subquery> }**
BETWEEN Predicate (range)	**<expression1> [NOT] BETWEEN <expression2> AND <expression3>**
IN Predicate	**<expression> [NOT] IN { <expression_list> | <subquery> }**
EXISTS Predicate	**[NOT] EXISTS (<subquery>)**
LIKE Predicate	**<expression1> [NOT] LIKE <expression2> [ESCAPE <expression3>]**
NULL Predicate	**<expression> IS [NOT] NULL**

Figure 180: SQL Predicates

A predicate is specified by combining one or more expressions or logical operators and returns one of the following logical or truth values:

`TRUE`, `FALSE`, or `UNKNOWN`.

The LIKE Predicate

- The `LIKE` predicate is used for string comparisons. A value, expression1, is tested for a pattern, expression2. Wildcard characters (%) and (_) may be used in the comparison string expression2. `LIKE` returns true if the pattern specified by expression2 is found.
- The percentage sign (%) matches zero or more characters and underscore (_) matches exactly one character. To match a percent sign or underscore in the `LIKE` predicate, an escape character must be used.
- Using the optional argument *ESCAPE expression3*, you can specify the escape character that will be used so that the underscore (_) or percentage sign (%) can be matched.

Operators

Unary	operator operand	unary plus operator(+) unary negation operator(-) logical negation(NOT)
Binary	operand1 operator operand2	multiplicative (*, /), additive (+,-) comparison operators (=,!=,<,>,<=,>=) logical operators (AND, OR)
Arithmetic Operators	-< expression > < expression > operator < expression >	Negation, Addition, Subtraction Multiplication, Division
String Operator	< expression > || < expression >	String concatenation
Comparison Operators	<expression> operator <expression>	>= Greater or equal to <= Less than or equal to !=, <> Not equal
Logical Operators	Search conditions can be combined using AND or OR operators. You can also negate them using the NOT operator.	AND, OR NOT
Set Operators	Set operators perform operations on the results of two or more queries.	UNION, UNION ALL, INTERSECT, EXCEPT

Figure 181: Operators

You can perform operations in expressions by using operators. Operators can be used for calculation, value comparison, or to assign values.

Functions

```
-- Find a list of SAP HANA courses
-- for the next 4 weeks (28 days)

SELECT name, description, cost, date
FROM courses
WHERE company = 'SAP'
      AND description LIKE '%HANA%'
AND DAYS_BETWEEN(CURRENT_DATE(),
Date) <= 28 AS in_x_days;
```

Figure 182: An Example of SQL Functions

Table 12: Functions

Classification of the Functions	Examples
Data type conversion	`CAST, TO_APLHANUM, TO_BIGINT, …`
Date and Time	`ADD_DAYS, ADD_MONTHS, ADD_YEARS, DAYS_BETWEEN, DAYNAME, CURRENT_DATE, …`

Classification of the Functions	Examples
Number	`ABS, ACOS, ASIN, ATAN, COS, SIN, TAN, …`
String	`CONCAT, LEFT, LENGTH, TRIM, …`
Miscellaneous	`IFNULL, CURRENT_SCHEMA, …`

Functions provide a reusable method to evaluate expressions and return information from the database. They are allowed anywhere an expression is allowed. Functions use the same syntax conventions used by SQL statements.

- Data type conversion functions

  Data type conversion functions are used to convert arguments from one data type to another, or to test whether they can be converted.

- Number Functions

  Number functions take numeric values or strings with numeric characters as inputs and returns numeric values. When strings with numeric characters are given as inputs, implicit conversion from string to number is performed automatically before computing the result values.

Expressions

```
-- Find out how many unique SAP HANA courses
-- are available

SELECT COUNT(DISTINCT name) as num_courses
FROM courses
WHERE company = 'SAP'
AND description LIKE '%HANA%';
```

Figure 183: An Example of an SQL Expression Using COUNT

Table 13: Expressions

Expression Type	Description
Case Expressions	`IF … THEN … ELSE` logic without using procedures in SQL statements.
Function Expressions	SQL built-in functions can be used as an expression.

Expression Type	Description
Aggregate Expressions	Uses an aggregate function to calculate a single value from the values of multiple rows in a column.
Subqueries in expressions	`SELECT` statement enclosed in parentheses.

An expression is a clause that can be evaluated to return values.

- Case Expressions

  A case expression allows the user to use `IF ... THEN ... ELSE` logic without using procedures in SQL statements.

- Function Expressions

  SQL built-in functions can be used as an expression.

- Aggregate Expressions

  An aggregate expression uses an aggregate function to calculate a single value from the values of multiple rows in a column.

- Subqueries in expressions

  A subquery is a `SELECT` statement enclosed in parentheses. The `SELECT` statement can contain only one select list item. When used as an expression, a scalar subquery is allowed to return only zero or one value.

Expressions: Examples

Case expression

You can use IF ... THEN ... ELSE logic without using procedures in SQL statements.

```
<expression> ::= CASE <expression>
                 WHEN <expression>
                 THEN <expression>, ...
                 [ ELSE <expression>]
                 { END | END CASE }
```

Aggregate Expressions

```
<aggregate_expression> ::= COUNT(*) | <agg_name> ( [ ALL | DISTINCT ] <expression>)

<agg_name> ::= COUNT | MIN | MAX | SUM | AVG | STDDEV | VAR
```

Figure 184: Expressions: Examples

Case Expression

If the expression following the `CASE` statement is equal to the expression following the `WHEN` statement, then the expression following the `THEN` statement is returned. Otherwise the expression following the `ELSE` statement is returned, if it exists.

Aggregate Expression

- COUNT - Counts the number of rows returned by a query. COUNT(*) returns the number of rows, regardless of the value of those rows and including duplicate values, whereas COUNT(<expression>) returns the number of non-NULL values for that expression returned by the query.
- MIN - Returns the minimum value of an expression.
- MAX - Returns the maximum value of an expression.
- SUM - Returns the sum of an expression.
- AVG - Returns the arithmetical mean of an expression.
- STDDEV - Returns the standard deviation of a given expression as the square root of the VARIANCE function.
- VAR - Returns the variance of an expression as the square of a standard deviation.

Creating Tables

```
CREATE [<table_type>] TABLE <table_name> <table_contents_source> ;

table_type ::=        COLUMN | ROW | HISTORY COLUMN | GLOBAL TEMPORARY | LOCAL
                      TEMPORARY

table_contents_source ::= ( <table_element>, ... )|[ (column_name, ...) ] |
                          [<like_table_clause> | <as_table_subquery>]
                          [ WITH [NO] DATA ] ]

table_element ::=        column_definition column_constraint | table_constraint
                         ( column_name, ... )

like_table_clause ::= LIKE like_table_name

as_table_subquery ::= AS (<select_query>)

column_definition ::= column_name data type [<column store data type>]
                      [<ddic data type>] [DEFAULT default_value]
                      [GENERATED ALWAYS AS <expression> ]
```

Figure 185: SQL statement: Create Table

Table Types

COLUMN	COLUMN-based storage should be used, if the majority of access is through a large number of tuples but with only a few selected attributes.
ROW	ROW-based storage is preferable, if the majority of access involves selecting a few records with all attributes selected.
HISTORY COLUMN	Creates a table with a particular transaction session type called *HISTORY*. Tables with session type HISTORY support **time travel**; the execution of queries against historic states of the database is possible.
GLOBAL TEMPORARY	Table definition is globally available while data is visible only to the current session. The table is visible in the catalog and is dropped at the end of the session.
LOCAL TEMPORARY	The table definition and data is visible only to the current session. The table is not visible in the catalog and is dropped at the end of the session. Table name must begin with the hash (#) symbol.

Figure 186: Create Table - Table Types

The history column table is particularly interesting because this allows SAP HANA to store all previous values of a record. Instead of updating a record when you edit it and thereby losing the previous values, SAP HANA keeps the previous record if the table type is a history column table. It makes the previous record with a `valid_to` time and date field, and inserts the record with the new values with a `valid_from` date and time field.

This allows you to go back in time and see what the table looked like at a specific point in the past. In future, you can expect to see many more features in the business systems where SAP will leverage this functionality, for example, for slowly changing dimensions or month-end processes.

```
-- Show the SAP HANA courses that
-- were available on January 1st, 2015

SELECT name, description, cost, date
FROM courses_history
WHERE company = 'SAP'
      AND description LIKE '%HANA%'
AS OF UTCTIMESTAMP '2015-01-01
12:00:00';
```

Figure 187: Use a History Column Table

Syntax Elements

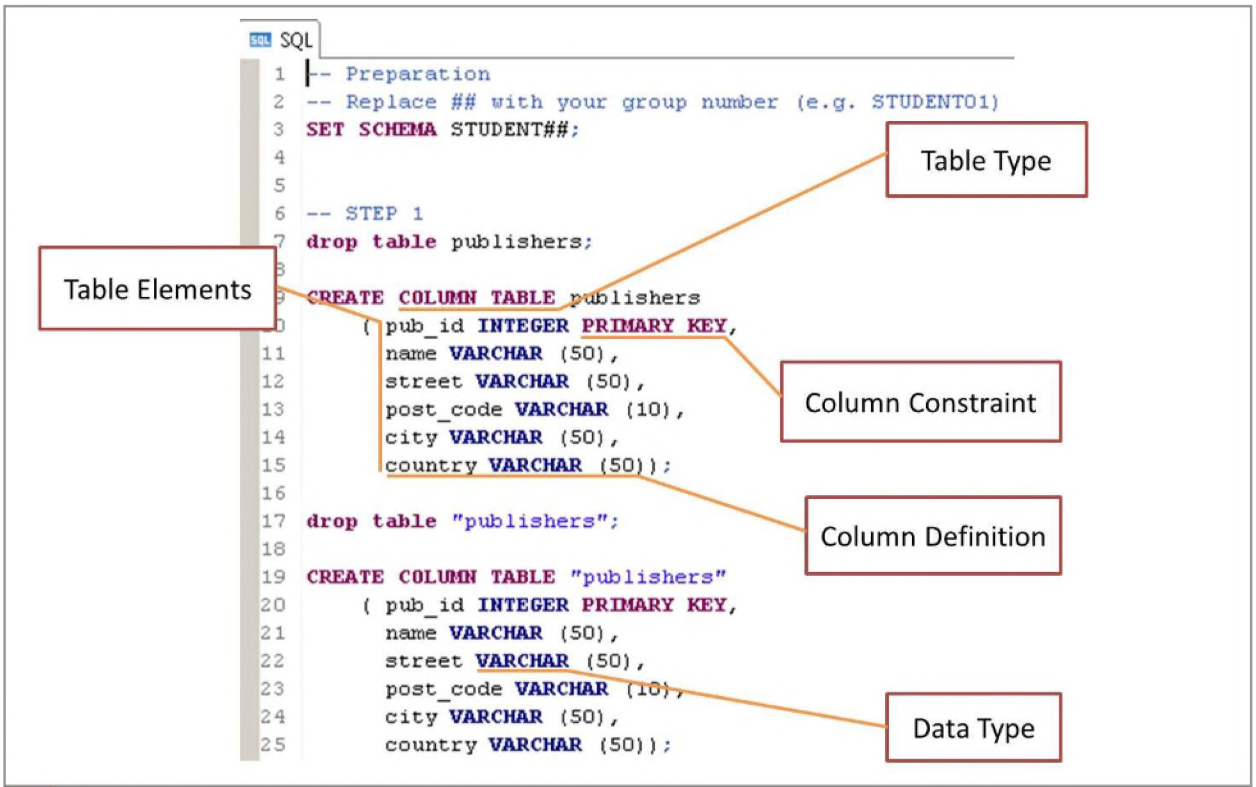


Figure 188: Create Table - Syntax Elements

column_constraint ::= NULL | NOT NULL | UNIQUE [BTREE | CPBTREE] | PRIMARY KEY [BTREE | CPBTREE]

table_constraint ::=UNIQUE [BTREE | CPBTREE] | PRIMARY KEY [BTREE | CPBTREE]

SQL Statement: Insert

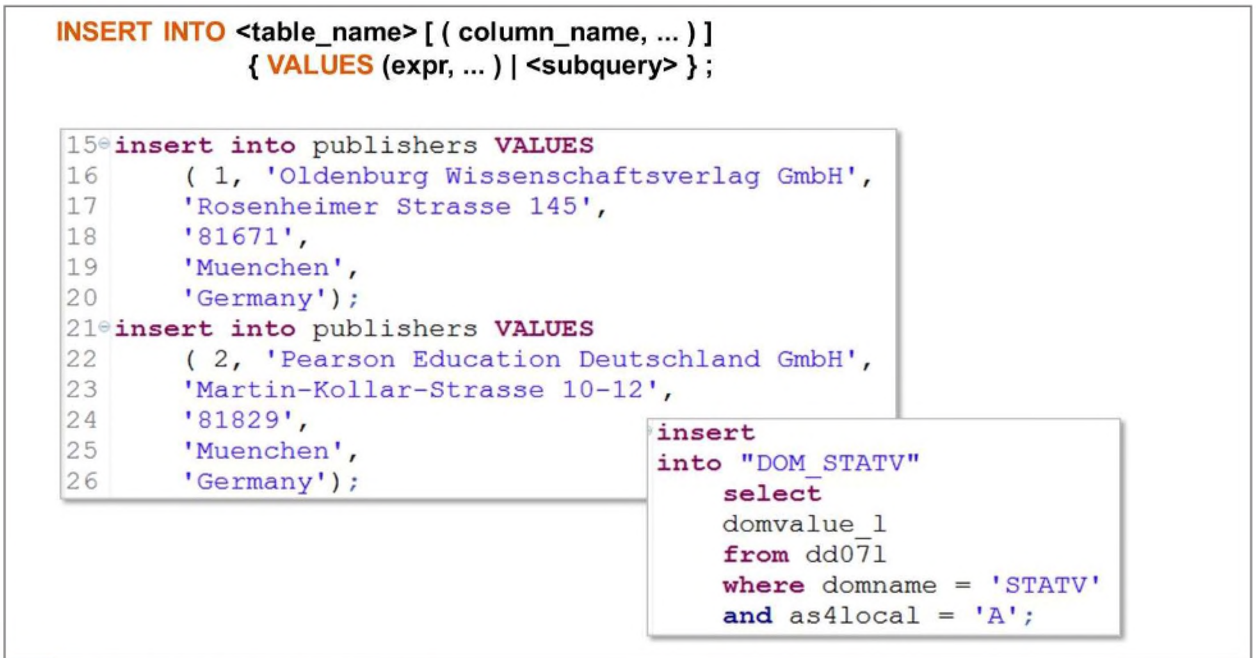


Figure 189: SQL statement: Insert

SQL Statement: Select

```
SELECT [TOP number] [ALL | DISTINCT]
<select_list>
<from_clause>
[<where_clause>]
[<group_by_clause>]
[<having_clause>]
[<order_by_clause>]
[<limit_clause>]
[<for update_clause>]
[<time_travel_clause>];
```

Anything in square brackets means these are optional clauses of the `SELECT` statement.

- <for_update_clause> ::= FOR UPDATE [OF <column_name>,]

  The `FOR UPDATE` keyword locks the selected rows so that other users cannot lock or update the rows until end of this transaction.

- <time_travel_clause> ::= AS OF [COMMIT ID|TIMESTAMP] [<commit_id> | <timestamp>]

  Can be used for statement level time travel to go back to the snapshot specified by `commit_id` or `timestamp`.

Unit 3
Exercise 15

Work with SQL Statements

Exercise Objectives

After completing this exercise, you will be able to:

- Create tables using SQL statements
- Insert values into tables using SQL Statements

Business Example

You are working with a book publisher, and for testing purposes, you need to create some sample tables including data.

Task 1: Create New Tables using SQL Statements

1. In the *Systems* view, open an SQL Console for your SAP HANA System.
2. From your *Student* folder, open the SQL file *HA300* → *002_CREATE_TABLE.sql*, and copy all its content to the SQL Console.

   If credentials are required to access this shared folder, use the following ones:

Field	Value
Network Share User	**hanastudent**
Network Share Password	**hanareadonly**

3. In the SQL editor, replace ## in all occurrences of *STUDENT##* with your group number and execute the SQL statement.

   > Caution:
   > Ensure that you have adjusted all occurrences of *STUDENT##* in the script before you proceed.

   What errors show up and why?

   ____

4. The statement will create two tables.

   Fill the table below with the exact table name and table type (Column / Row) of the newly created tables:

Table Identifier	Table Type
Table 1:	
Table 2:	

5. Refresh the table node in your schema *STUDENT##* and check the identifiers (table names) of the newly created tables.

   Why is the second table created with small letters?

   ____

Task 2: Create a Table from a Template without and with Data

1. In your SQL query, remove the block comment marks (`/*` and `*/`) in the SQL code for STEP 2, and execute again.

   Do not change the comment `-- WITH DATA;` for the first execution.

2. Check if the newly created table *CurrDecimals* contains data.

> Hint:
> Open the table in preview mode from the *Systems* view.

3. Modify the SQL statement (step 2) to include data in the new table you create, and execute the SQL statement again.

   You need to remove the two dashes in the comment `-- WITH DATA;`.

4. Check if the table *CurrDecimals* now contains data.

> Hint:
> If the table *CurrDecimals* is still open in preview mode, you just need to refresh the content display.

5. Delete the two tables *"publishers"* and *"CurrDecimals"* from your *STUDENT##* schema.

   You do not need these tables for the following steps.

> Hint:
> To delete a table, you can use the context menu of the table in the *Systems* view.
>
> Selecting *Cascade* and *Delete Catalog Object* in the deletion confirmation dialog box will completely delete the table and any dependent objects.

6. Clear your SQL Console before proceeding with the next activities.

Task 3: Insert Values using SQL Statements

1. In the *Systems* view, Open a SQL Console for your SAP HANA System.

2. From your *Student* folder, open the SQL file *HA300 → 003_INSERT_VALUES.sql*, and copy all its content to the SQL Console.

3. In the SQL editor, replace ## in all occurrences of *STUDENT##* with your group number and execute the SQL statement.

> Caution:
> Ensure that you have adjusted all occurrences of *STUDENT##* in the script, before you proceed.

4. Display the content of tables *PUBLISHERS* and *BOOKS*.

> Note:
> Make sure you see content in these tables, as you will need them in exercises following this one.

5. Clear your SQL Console before proceeding with the next activities.

Unit 3
Solution 15

Work with SQL Statements

Exercise Objectives

After completing this exercise, you will be able to:

- Create tables using SQL statements
- Insert values into tables using SQL Statements

Business Example

You are working with a book publisher, and for testing purposes, you need to create some sample tables including data.

Task 1: Create New Tables using SQL Statements

1. In the *Systems* view, open an SQL Console for your SAP HANA System.
   a) Right-click your system and choose *Open SQL Console*.

2. From your *Student* folder, open the SQL file *HA300 → 002_CREATE_TABLE.sql*, and copy all its content to the SQL Console.

   If credentials are required to access this shared folder, use the following ones:

Field	Value
Network Share User	**hanastudent**
Network Share Password	**hanareadonly**

   a) Choose *Start → Student - hanastudent - hanareadonly*.

   > Note:
   > If the Student shortcut is not pinned to your Windows *Start* page, use the *Search* tool (magnifier on the top-right corner or the *Start* page).

   b) Enter the provided credentials and choose *OK*, If needed.

   c) Open the file *HA300 → 002_CREATE_TABLE.sql*.

   d) Select all the content of the file (**Ctrl+A**) and copy it to the clipboard (press Ctrl+C).

   e) To paste the content in the SQL Console, press Shift+Ctrl+INS (a special shortcut is used in the SQL Console) or right-click anywhere inside the SQL Console and choose *Paste*.

```
-- Preparation
```


For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

```
-- First change STUDENT## to your group number
-- E.g. STUDENT01

SET SCHEMA STUDENT##;

-- STEP 1
drop table publishers;

CREATE COLUMN TABLE publishers
    ( pub_id INTEGER PRIMARY KEY,
      name VARCHAR (50),
      street VARCHAR (50),
      post_code VARCHAR (10),
      city VARCHAR (50),
      country VARCHAR (50));

drop table "publishers";

CREATE COLUMN TABLE "publishers"
    ( pub_id INTEGER PRIMARY KEY,
      name VARCHAR (50),
      street VARCHAR (50),
      post_code VARCHAR (10),
      city VARCHAR (50),
      country VARCHAR (50));

/*
-- STEP 2

drop table "CurrDecimals";

CREATE COLUMN TABLE "CurrDecimals" LIKE "TCUR"."TCURX" -- WITH DATA;

*/
```

3. In the SQL editor, replace ## in all occurrences of *STUDENT##* with your group number and execute the SQL statement.

> Caution:
> Ensure that you have adjusted all occurrences of *STUDENT##* in the script before you proceed.

What errors show up and why?

___

a) In the SQL editor, choose *Edit* → *Find/Replace* and replace ## with your group number.

b) To execute the script, press **F8** or choose *(Execute)*.

c) If you review the script, you will see that it tries to drop (remove) each of the two tables *publishers* and *"publishers"* before creating them.

   If the table does not already exist, that is, the first time you execute it in this exercise, you will see an error.

   You can try to run the same script another time to check that the error disappears.

4. The statement will create two tables.

Fill the table below with the exact table name and table type (Column / Row) of the newly created tables:

Table Identifier	Table Type
Table 1:	
Table 2:	

a) You need to find both table names and table types.

> Hint:
> To find the table names, in the *Modeler* perspective, refresh the table node of your schema in the *Systems* view (press F5). Right click the table identifier and choose *Open Definition* to check the table type.

5. Refresh the table node in your schema *STUDENT##* and check the identifiers (table names) of the newly created tables.

   Why is the second table created with small letters?

   ____

   a) To refresh the display, select your *STUDENT##* schema and press F5.

   b) There are two tables, one with a lowercase name and one with the name in capitals. All identifiers not included in double quotes are automatically translated into uppercase.

> Hint:
> Pay close attention to the double quotes in the SQL script.

Task 2: Create a Table from a Template without and with Data

1. In your SQL query, remove the block comment marks (`/*` and `*/`) in the SQL code for STEP 2, and execute again.

   Do not change the comment `-- WITH DATA;` for the first execution.

   a) In the SQL Editor, remove the two block comment marks (`/*` and `*/`).

   b) To execute the script, press F8 or choose *(Execute)*.

2. Check if the newly created table *CurrDecimals* contains data.

> Hint:
> Open the table in preview mode from the *Systems* view.

   a) In the *Systems* view, right-click the table *CurrDecimals* and choose *Open Data Preview*.

3. Modify the SQL statement (step 2) to include data in the new table you create, and execute the SQL statement again.

   You need to remove the two dashes in the comment `-- WITH DATA;`.

a) In the SQL Editor, remove the two dashes in the comment `-- WITH DATA;`.

> Note:
> Just remove the dashes, not the actual `WITH DATA;` command.

b) To execute the script, press F8 or choose *(Execute)*.

4. Check if the table *CurrDecimals* now contains data.

> Hint:
> If the table *CurrDecimals* is still open in preview mode, you just need to refresh the content display.

a) If the *Data Preview* for table *CurrDecimals* is still open, choose *(Refresh)*.

Alternatively, in the *Systems* view, right-click the table *CurrDecimals* and choose *Open Data Preview*.

> Hint:
> To drop a table by right clicking on the table in the *Systems* view, select *Delete* from the context menu. Select *Cascade* and *Delete Catalog Object* in the deletion confirmation window to completely delete the table and any dependent objects.

5. Delete the two tables *"publishers"* and *"CurrDecimals"* from your *STUDENT##* schema.

You do not need these tables for the following steps.

> Hint:
> To delete a table, you can use the context menu of the table in the *Systems* view.
>
> Selecting *Cascade* and *Delete Catalog Object* in the deletion confirmation dialog box will completely delete the table and any dependent objects.

a) In the *Systems* view, right-click the table *STUDENT## → Tables → publishers* object and choose *Delete*.

b) In the *Delete* dialog box, check select the *Cascade* checkbox and choose *OK*.

c) Repeat the previous steps for the other table, *CurrDecimals*.

d) Close the data preview window.

6. Clear your SQL Console before proceeding with the next activities.

a) Right-click anywhere in your SQL Console and choose *Clear*.

Task 3: Insert Values using SQL Statements

1. In the *Systems* view, Open a SQL Console for your SAP HANA System.

a) Right-click your system and choose *Open SQL Console*.

2. From your *Student* folder, open the SQL file *HA300 → 003_INSERT_VALUES.sql*, and copy all its content to the SQL Console.

   a) From Windows Explorer, in your *student* folder, open the file *HA300 → 003_INSERT_VALUES.sql*.

   b) Select all the content of the file (**Ctrl+A**) and copy it to the clipboard (press **Ctrl+C**).

   c) To paste the content in the SQL Console, press **Shift+Ctrl+INS** (a special shortcut is used in the SQL Console) or right-click anywhere inside the SQL Console and choose *Paste*.

```
-- First change STUDENT## to your group number
-- E.g. STUDENT01

SET SCHEMA STUDENT##;


 --test data
DROP TABLE publishers;
CREATE COLUMN TABLE publishers
    ( pub_id INTEGER PRIMARY KEY,
      name VARCHAR(50),
      street VARCHAR(50),
      post_code VARCHAR(10),
      city VARCHAR(50),
      country VARCHAR(50)
     );
insert into publishers VALUES
    ( 1, 'Oldenburg Wissenschaftsverlag GmbH',
    'Rosenheimer Strasse 145',
    '81671',
    'Muenchen',
    'Germany');
insert into publishers VALUES
    ( 2, 'Pearson Education Deutschland GmbH',
    'Martin-Kollar-Strasse 10-12',
    '81829',
    'Muenchen',
    'Germany');
insert into publishers VALUES
    ( 3, 'mitp & bhv-Buch', 'Augustinusstrasse 9d', '50226',
'Frechen', 'Germany');
insert into publishers VALUES
    ( 4, 'Roof Music', 'Prinz-Regent-Strasse 50-60', '44795',
'Bochum', 'Germany');

-- Additional table for subsequent exercises
drop TABLE books;
CREATE COLUMN TABLE books( isbn VARCHAR(20) PRIMARY KEY, title
VARCHAR(50), publisher INTEGER, edition INTEGER, year VARCHAR(4),
price DECIMAL(5, 2), crcy VARCHAR(3));

insert into books VALUES ('978-3-486-57690-0', 'Datenbanksysteme: Eine
Einfuehrung', 1, 6, '2006', '39.80', 'EUR');
insert into books VALUES ('978-3-86894-012-1', 'Grundlagen von
Datenbanken', 2, 3, '2009', '29.95', 'EUR');
insert into books VALUES ('978-3-8266-1664-8', 'Datenbanken: Konzepte
und Sprachen', 3, 3, '2008', '39.95', 'EUR');
insert into books VALUES ('978-3-486-59002-9', 'Algorithmen: Eine
Einführung', 1, 3, '2010', '79.80', 'EUR');
```

3. In the SQL editor, replace ## in all occurrences of *STUDENT##* with your group number and execute the SQL statement.

> Caution:
> Ensure that you have adjusted all occurrences of *STUDENT##* in the script, before you proceed.

a) In the SQL editor, choose *Edit → Find/Replace* and replace ## with your group number.

b) To execute the script, press F8 or choose *(Execute)*.

4. Display the content of tables *PUBLISHERS* and *BOOKS*.

> Note:
> Make sure you see content in these tables, as you will need them in exercises following this one.

a) Right-click the created tables in the *Systems* view and choose *Open Data Preview*. The *Raw Data* tab shows you a preview of the content of the table.

b) Close the Data Preview windows.

5. Clear your SQL Console before proceeding with the next activities.

a) Right-click anywhere in your SQL Console and choose *Clear*.

LESSON SUMMARY

You should now be able to:

- Describe SAP HANA SQL

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

Unit 3
Lesson 2

Working with SQLScript

LESSON OVERVIEW

This lesson will give you a general introduction to SQLScript. You will learn about the benefits of SQLScript and how you can use it in the context of scripted Calculation views, table functions, or procedures.

The procedures will be covered in a dedicated lesson later on.

Business Example

You have created graphical Calculation Views, but due to your business requirements, you find it necessary to expand information models by using script-based information models.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Work with SQLScript
- Explain the SQLScript implementation logic

Introducing SQLScript

SQLScript is a collection of extensions to Structured Query Language (SQL). The main motivation for SQLScript is to embed data intensive application logic into the database, in contrast to a traditional approach where the application logic is normally executed in an application server.

SQLScript exposes many of the in-memory features of SAP HANA to the SQL developers. Column tables, parameterized information views, delta buffers, working with multiple result sets in parallel, built-in currency conversions at database level, fuzzy text searching, spatial data types, and predictive analysis libraries makes SAP HANA unique. The only way to make this available via SQL queries, even from other applications, is to provide it as SQL extensions.

In traditional client-server approaches, the business logic is executed in the application server. With SAP HANA, much of this logic and executions is pushed down into the SAP HANA database. This approach is very different from the standard SQL way of working. SQLScript caters for these requirements.

One of the unique approaches in SQLScript is to use variables to break a large complex SQL statement into smaller, simpler statements. This makes the code much easier to understand, and it also helps with SQL HANA's performance, because many of these smaller statements can be run in parallel.

Let's have a look at an example.

```
books_per_publisher = SELECT publisher, COUNT (*) AS num_books FROM
BOOKS GROUP BY publisher;
publisher_with_most_books = SELECT * FROM :books_per_publisher WHERE
num_books >= (SELECT MAX (num_books) FROM :books_per_publisher);
```

We would normally write this as a single SQL statement using a temporary table or by repeating a sub-query multiple times. We have broken this into two smaller SQL statements by using table variable.

The first statement calculates the number of books each publisher has and stores the entire result set into the table variable called `books_per_publisher`.

This variable, containing the entire result set, is used twice in the second statement.

Notice that the table variable is prepended in SQLScript with a colon (':') to indicate that this is used as an input variable. All output variables just have the name, and all input variables have a colon prepended.

The second statement uses `:books_per_publisher` as inputs, and uses a nested `SELECT`.

The SQLScript compiler and optimizer will determine how to best execute these statements, whether by using a common sub-query expression with database hints or by combining this into a single complex query. The code becomes easier to write and understand, more maintainable, and developer productivity increases.

By breaking the SQL into smaller statements, we also mirror the way in which we have learned to build our graphical information views in SAP HANA. Just like we start building graphical information views in layers, starting from the bottom up, we do the same with our SQLScript code.

Whatever you created in graphical calculation views, you can also create using SQLScript.

An example would be a union node in the graphical calculation views. You can also create unions with SQLScript, using the `UNION` operator.

```
SELECT author, title, price FROM BOOKS
UNION
SELECT author, title, price FROM BOOKS_OUT_OF_PRINT;
```

> Note:
> We can also use the `UNION ALL` operator instead of `UNION`. The `UNION` operator returns only distinct values, whereas the `UNION ALL` will also return all duplicate records.

- **SQLScript** extends ANSI-92 SQL. Just like other database vendors extend SQL in their databases, SAP HANA extends SQL using SQLScript.
- **SQLScript** enables you to access SAP HANA-specific features like column tables, parameterized information views, delta buffers, working with multiple result sets in parallel, built-in currency conversions at database level, fuzzy text searching, spatial data types, and predictive analysis libraries.
- **SQLScript** allows developers to push data intensive logic into the database
- **SQLScript** encourages developers to maintain algorithms using a set-oriented paradigm, instead of a one record at a time paradigm
- **SQLScript** does however allow looping through results and using if-then-else logic
- **SQLScript** allows you to break complex queries into multiple smaller statements, thereby simplifying your SQL coding, and enhancing parallelism via the optimizer.

Figure 190: SQLScript: Concept

SQLScript Data Types

SQLScript has additional data types to help address missing features from standard SQL, for example, spatial data types in SAP HANA.

Classification	Standard ANSI-92 Data Types	SQLScript-specific Data Types
Date/Time	DATE, TIME, TIMESTAMP	SECONDDATE
Numeric	DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, DOUBLE	TINYINT, BIGINT, SMALLDECIMAL
Character string	CHAR, NCHAR, VARCHAR, NVARCHAR	ALPHANUM, SHORTTEXT
Binary	BINARY, VARBINARY	
Large Object		BLOB, CLOB, NCLOB, TEXT, BINTEXT
Boolean		BOOLEAN
Spatial		ST_POINT, ST_GEOMETRY

Figure 191: ANSI-92 and SQLScript Data Types

Table Type Extension

Table Type:

SQLScript's datatype extension also allows the definition of table types. These table types are used to define parameters for a procedure.

A table type is created using the **CREATE TYPE** and delete using **DROP TYPE** statement.

Syntax:

CREATE TYPE [schema.]name AS TABLE (name1 type1 [, name2 type2,...])

DROP TYPE [schema.]name [CASCADE]

Figure 192: SQLScript: Data Type Extensions

SQLScript also allows the definition of table types. These table types are used to define parameters for a procedure that represents tabular results.

Using Variables

The use of variables in SQL statements is not specified by the ANSI-92 SQL. In SAP HANA, SQLScript implements the concept of variable as an extension to ANSI-92 SQL.

Variables can be declared, assigned, and reused within a script.

Here is a quick overview on how variables are used within SQLScript:

Using Variables in SQLScript

- Declare a variable

  ```
  DECLARE <variable_name> <type> [NOT NULL] [= <value>]
  ```

  Example 1: `DECLARE a int;`

  Example 2: `DECLARE b int = 0;` (the value is assigned when creating the variable)

- Assign a variable (define the value of a variable)

  ```
  <variable_name> = <value> or <expression>
  ```

  Example 1: `a = 3;`

  Example 2: `b = select count(*) from baseTable;`

- Use a variable in an expression

  `:<variable_name>` (with a colon) returns the current value of the variable

  Example: `b = :a + 5;` (assigns the value of a + 5 to b)

Note:
The examples above are simplified and do not cover the complete SQLScript syntax for variables. You can find more information in the SAP HANA SQLScript Reference guide, available at http://help.sap.com/hana

Imperative Logic

Occasionally, we might prefer to receive the answers from a database query one record at a time. This is most often the case when running SQL queries from application servers. In these cases, we will use imperative logic, meaning we will loop through the results one at a time. We can also use conditionals like `If-then` logic.

This is (mostly) not available in ANSI SQL, and therefore SAP HANA provides this as part of SQLScript.

> Hint:
> Remember that imperative logic cannot have the same performance as declarative logic. If you require the best performance, try to avoid imperative logic.

There are different ways to implement imperative logic in SQLScript.

Control Statements and Dynamic SQL

```
IF <bool-expr1>
THEN
  {then-stmts1}
{ELSEIF <bool-expr2>
THEN
  {then-stmts2}}
{ELSE
  {else-stmts3}}
END IF
```

Example:

```
SELECT count(*) INTO found
FROM books WHERE isbn = :v_isbn;

IF :found IS NULL THEN

CALL ins_msg_proc
('result of count(*) cannot be NULL');

ELSE

CALL ins_msg_proc
('result of count(*) not NULL - as expected');

END IF;
```

Figure 193: The IF Statement

The `IF` statement consists of a Boolean expression `bool-expr1`. If this expression evaluates to true then the statement `then-stmts1` in the mandatory `THEN` block is executed. The `IF` statement ends with `END IF`. The remaining parts are optional.

If the Boolean expression `bool-expr1` does not evaluate to true the `ELSE` branch is evaluated. In most cases this branch starts with `ELSE`. The statement `else-stmts3` is executed without further checks. After an else branch no further `ELSE` branch or `ELSEIF` branch is allowed.

Alternatively, when `ELSEIF` is used instead of `ELSE` another Boolean expression `bool-expr2` is evaluated. If it evaluates to true, the statement `then-stmts2` is executed. In this fashion an arbitrary number of `ELSEIF` clauses can be added.

This statement can be used to simulate the switch-case statement known from many programming languages.

WHILE and FOR Loops

```
WHILE <bool-stmt> DO
  {stmts}
END WHILE

FOR <loop-var> IN {REVERSE} <start> .. <end> DO
  {stmts}
END FOR
```

Figure 194: WHILE and FOR Loops

- The WHILE Loop

  The `WHILE` loop executes the statement `stmts` in the body of the loop as long as the Boolean expression at the beginning `bool-stmt` of the loop evaluates as true.

- The FOR Loop

  The FOR LOOP iterates a range of numeric values – denoted by start and end in the syntax – and binds the current value to a variable (`loop-var`) in ascending order. Iteration starts with value start and is incremented by one until the `loop-var` is larger than end.

  Therefore, if start is larger than end, the body loop will not be evaluated. For each enumerated value of the loop variable the statements in the body of the loop are evaluated. The optional keyword `REVERSE` specifies to iterate the range in descending order.

The EXEC Statement

The `EXEC` statement is used to help create dynamic SQL statements. Let's illustrate this with an example. You want to get the travel arrangements for users, and a web form is requesting the user's passport number, insert this passport number into a partially prepared SQL statement, and then execute this SQL statement using the `EXEC` statement to get the travel plans.

```
EXEC '<SQL statement>'
```

Dynamic SQL statements are used to construct SQL statements at runtime in a procedure.

The `EXEC` statement executes the SQL statement passed in a string argument. This statement allows for dynamically constructing an SQL statement at execution time of a procedure.

For example, a `SELECT` statement could reference a table whose name is not known until runtime of the procedure. Perhaps the table name is retrieved from a lookup based on a condition.

Therefore, on the positive side, dynamic SQL allows the use of variables where they might not be supported in SQLScript or more flexibility in creating SQL statements.

On the negative side, dynamic SQL comes with additional costs at runtime:

- Opportunities for optimizations are limited.
- The statement is potentially recompiled every time the statement is executed.
- Not possible to use SQLScript variables in the SQL statement (but when constructing the SQL statement string).
- Not possible to bind the result of a dynamic SQL statement to a SQLScript variable.
- Be very careful to avoid SQL injection bugs that might harm the integrity or security of the database.

  However, SAP HANA SPS11 brings a new feature to secure Dynamic SQL

> Note:
> It is generally recommended to avoid dynamic SQL because it might have a negative impact on security or performance.

Securing Dynamic SQL

SAP HANA SPS11 comes with three built-in procedures that can be used to help preventing SQL injection attacks.

Built-In Procedures to Secure Dynamic SQL

- `ESCAPE_SINGLE_QUOTES (string_var)`

  Used for variables containing a SQL string literal
- `ESCAPE_DOUBLE_QUOTES (string_var)`

  Used for variables containing a delimited SQL identifier
- `IS_SQL_INJECTION_SAFE(string_var[, num_tokens])`

  Used to check that a variable contains safe simple SQL identifiers (up to num_tokens, default is 1)

You can find details on these new procedures in the SAP HANA SQLSript Reference guide, available at http://help.sap.com/hana.

User Defined Function

Table and Scalar User Defined Functions (UDF) can be created in two different ways:

- By executing a `CREATE FUNCTION` statement within the SQL Editor

  You create a UDF that is a pure run-time object and as such is subject to some limitations:
  - It cannot be transported via a Delivery Unit
  - It cannot be used as data source in a Graphical Calculation View
- By creating a design-time object in a package, and activating it to generate the corresponding runtime object.

  This is the recommended way to proceed. The design-time object is a Core Data Services (CDS) document with extension `.hdbfunction` (scalar) or `.hdbtablefunction` (tabular).

> Note:
> You can get a brief overview of Core Data Services in the lesson *Building a Virtual Data Model with CDS Views*

Table Functions

Table Functions are now the recommended script-based artefact whenever a modeling requirement cannot be fulfilled with graphical information views.

In previous versions of SAP HANA, it was possible to create scripted calculation views, which are similar in their key principles but provide less flexibility and performance. In particular, Table User Defined Functions allow an improved movement of filters from client tools down to the source data.

You create design-time Table Functions with the *SAP HANA Developer* perspective.

The corresponding run-time objects are created and owned by the *_SYS_REPO* technical user, and you can find them under the *_SYS_BIC* schema, in the *Functions* folder.

It is also possible to convert existing Scripted Calculation Views into Table Functions, with a migration tool available in SAP HANA Studio since SAP HANA SPS11.

Table User Defined Functions (or Table Functions) have replaced Scripted Calculation Views

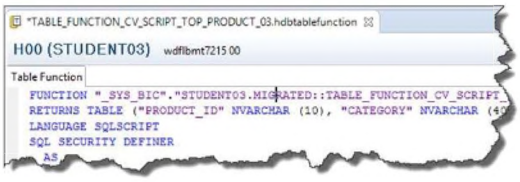


Table Functions

- **are side-effect free read-only functions that use standard SQL**
- **can be used as source data in Graphical Calculation Views**
- **consume data from tables, models, functions, predictive algorithms, etc.**

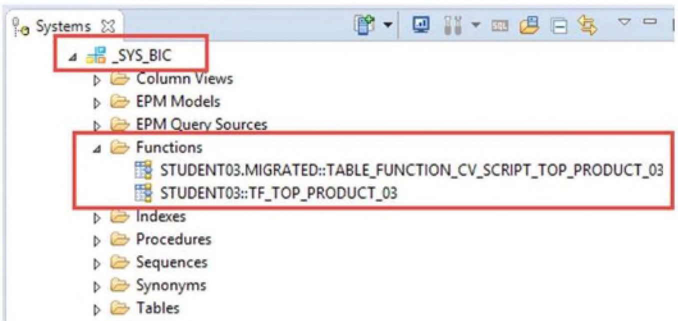


Figure 195: Table Functions

> Note:
> Table functions are read-only functions. This means that you cannot use `INSERT`, `UPDATE`, `DELETE`, `COMMIT`, or `EXEC SQL` statement inside a table function. If you have any of these SQL statements in a table function, it will not activate.

Create a Table Function and Use It in Graphical Calculation Views

Exercise Objectives

After completing this exercise, you will be able to:

- Create a Table User Defined Function (UDF)
- Use the Table UDF in a Graphical Calculation View of type Dimension
- Map the input of the Table Function to an input parameter in the Graphical Calculation View
- Use the Table UDF in a Graphical Calculation View of type Cube

Business Example

Your calculation view needs to find the name of an employee, even when the end-user searching for that employee types the surname of the employee incorrectly. You realize that the only way you can call such search functionality is using a user-defined function (UDF), so you create a table function.

You then use an input parameter to ask the end-users what employee they are looking for in the graphical calculation view. You map the input parameter's value to the table functions input. The table function searches the person using the built-in fuzzy text search of SAP HANA.

Once the correct employee is found, SAP HANA then uses this to show the details of that employee, or even the sales made by that employee.

Task 1: Define the SAP HANA Studio Preferences

You need to change some validation rules for the table functions. These validation rules are not used any more by the web-based development environments. So you can also safely switch them off in the SAP HANA studio.

1. Change the validation rule preferences to not give a warning when using *Table Functions*.

   The following rules need to be deactivated:

   - *Name and Description → ABAP Name Check*
   - *Content → Description Check*
   - *Content → Naming Convention Check*

Task 2: Create the Table Function

You want to find the name of an employee, even when you type the surname of the employee incorrectly. You realize that the only way you can call such search functionality is using a user-defined function (UDF). You create a table function.

1. Change the SAP HANA studio perspective to the SAP HANA Development perspective.

2. Edit the code of your table function with the following code. You can find it in your *Student* folder, in the file *ha300 → Employees Search Table Function.sql*.

```
FUNCTION "STUDENT##"."STUDENT##::TF_FUZZY_EMPLOYEES" (lastNameFilter
NVARCHAR(40))
    RETURNS table ( EMPLOYEEID NVARCHAR(10),
       "NAME.FIRST" NVARCHAR(40),
       "NAME.LAST" NVARCHAR(40),
       EMAILADDRESS NVARCHAR(255),
       ADDRESSID NVARCHAR(10), CITY NVARCHAR(40),
       POSTALCODE NVARCHAR(10), STREET NVARCHAR(60))
    LANGUAGE SQLSCRIPT
    SQL SECURITY INVOKER AS
BEGIN

RETURN
  select a."EMPLOYEEID", a."NAME.FIRST",
         a."NAME.LAST", a."EMAILADDRESS",
         a."ADDRESSES.ADDRESSID" as "ADDRESSID",
         b."CITY", b."POSTALCODE", b."STREET"
  from "sap.hana.democontent.epm.data::MD.Employees" as a
  inner join
  "sap.hana.democontent.epm.data::MD.Addresses" as b
  on a."ADDRESSES.ADDRESSID" = b.ADDRESSID
  where contains("NAME.LAST", :lastNameFilter, FUZZY(0.7));
END;
```

Note:
Notice the *CONTAINS* and *FUZZY* statements in this code. This feature enables you to find the names of employees even if they are misspelled. You will learn more about this in Unit 5.

3. Activate your table function.

Task 3: Use the Table Function in a Dimension Calculation View

1. Create a dimension calculation view.
2. Map the inputs of the table function to an input parameter in your calculation view.
3. Use the table function in the calculation view to find employees.

Unit 3
Solution 16

Create a Table Function and Use It in Graphical Calculation Views

Exercise Objectives

After completing this exercise, you will be able to:

- Create a Table User Defined Function (UDF)
- Use the Table UDF in a Graphical Calculation View of type Dimension
- Map the input of the Table Function to an input parameter in the Graphical Calculation View
- Use the Table UDF in a Graphical Calculation View of type Cube

Business Example

Your calculation view needs to find the name of an employee, even when the end-user searching for that employee types the surname of the employee incorrectly. You realize that the only way you can call such search functionality is using a user-defined function (UDF), so you create a table function.

You then use an input parameter to ask the end-users what employee they are looking for in the graphical calculation view. You map the input parameter's value to the table functions input. The table function searches the person using the built-in fuzzy text search of SAP HANA.

Once the correct employee is found, SAP HANA then uses this to show the details of that employee, or even the sales made by that employee.

Task 1: Define the SAP HANA Studio Preferences

You need to change some validation rules for the table functions. These validation rules are not used any more by the web-based development environments. So you can also safely switch them off in the SAP HANA studio.

1. Change the validation rule preferences to not give a warning when using *Table Functions*.

   The following rules need to be deactivated:

   - *Name and Description → ABAP Name Check*
   - *Content → Description Check*
   - *Content → Naming Convention Check*

   a) Choose *Window → Preferences → SAP HANA → Modeler → Validation Rules*.

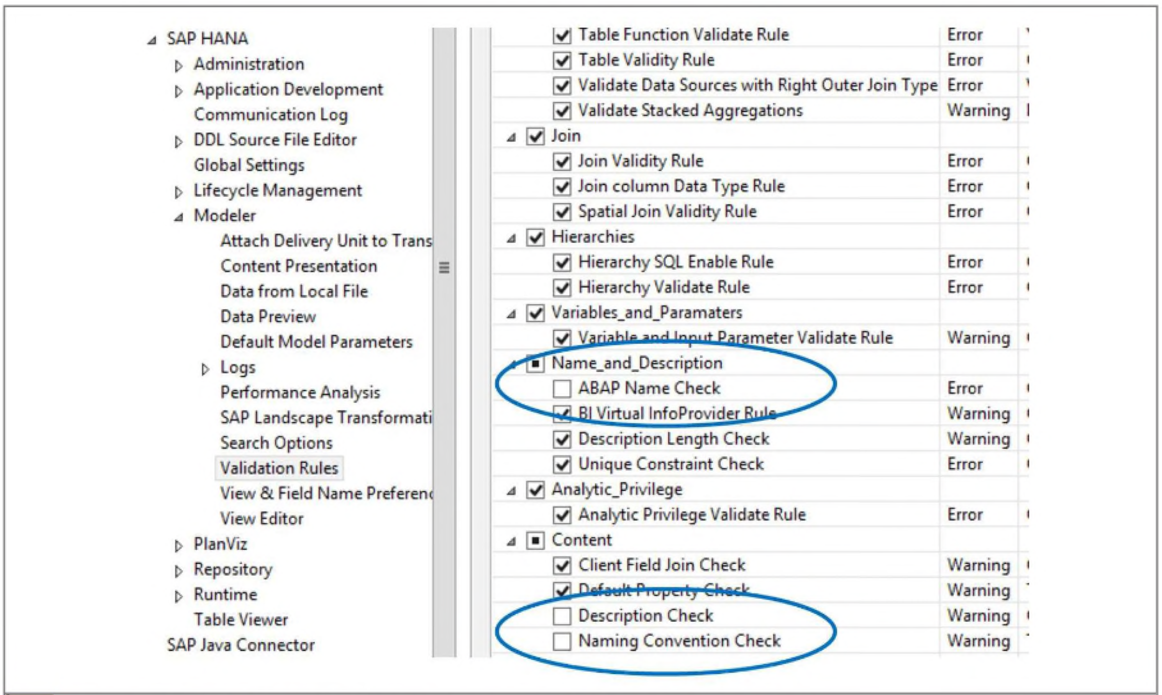


Figure 196: SAP HANA Studio Preferences for Table Function Validations

b) In the preferences tree, choose *SAP HANA* → *Modeler* → *Validation Rules* and find the node *Name_and_Description*. Deselect the *ABAP Name Check* checkbox.

c) Proceed similarly to deselect the two other validation rules.

d) Choose *OK*.

Task 2: Create the Table Function

You want to find the name of an employee, even when you type the surname of the employee incorrectly. You realize that the only way you can call such search functionality is using a user-defined function (UDF). You create a table function.

1. Change the SAP HANA studio perspective to the SAP HANA Development perspective.

   a) In the top right of the SAP HANA studio, the current perspectives for the SAP HANA studio are displayed. To get a list of available perspectives, choose the + icon .

   Choose the *SAP HANA Development* perspective.

   b) On the left side of the SAP HANA studio, look at the *Systems* area. You will notice that you now have two extra tabs called *Repositories* and *Project Explorer* available.

   c) Select the *Repositories* tab.

   d) Right-click the *H00* system name. In the menu, choose *Create Repository Workspace*.

   e) In the *Create New Repository Workspace* dialog, keep the default values and choose *Finish*.

   When you expand the workspace, you will see a list of the packages in your H00 system.

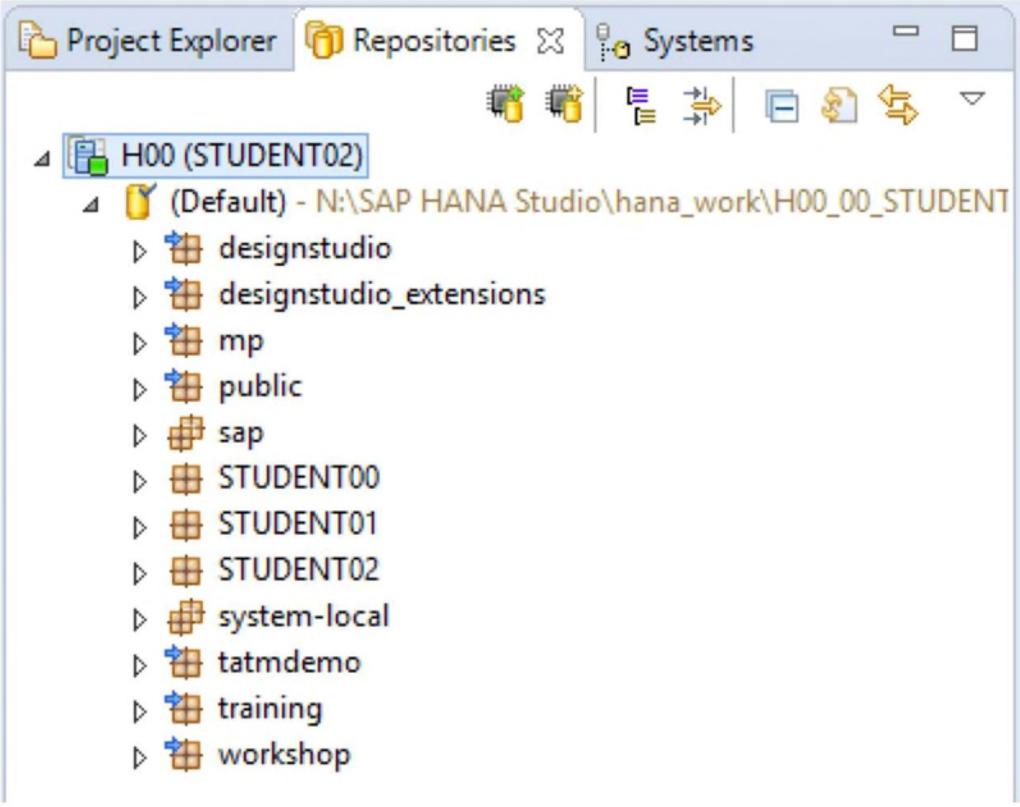


Figure 197: Create a New Repository Workspace

f) To create your table function in your package, expand your *STUDENT##* package (where ## is your student number).

g) Right-click your *STUDENT##* package. In the menu, choose *New → Other... → SAP HANA → Database Development → Table Function*. Choose *Next*.

h) In the *File name* field, enter **TF_FUZZY_EMPLOYEES**.
The SAP HANA studio will automatically make it a .hdbtablefunction file.

i) Choose *Finish*.

2. Edit the code of your table function with the following code. You can find it in your *Student* folder, in the file *ha300 → Employees Search Table Function.sql*.

```
FUNCTION "STUDENT##"."STUDENT##::TF_FUZZY_EMPLOYEES" (lastNameFilter
NVARCHAR(40))
    RETURNS table ( EMPLOYEEID NVARCHAR(10),
       "NAME.FIRST" NVARCHAR(40),
       "NAME.LAST" NVARCHAR(40),
       EMAILADDRESS NVARCHAR(255),
       ADDRESSID NVARCHAR(10), CITY NVARCHAR(40),
       POSTALCODE NVARCHAR(10), STREET NVARCHAR(60))
    LANGUAGE SQLSCRIPT
    SQL SECURITY INVOKER AS
BEGIN

RETURN
  select a."EMPLOYEEID", a."NAME.FIRST",
         a."NAME.LAST", a."EMAILADDRESS",
         a."ADDRESSES.ADDRESSID" as "ADDRESSID",
         b."CITY", b."POSTALCODE", b."STREET"
  from "sap.hana.democontent.epm.data::MD.Employees" as a
  inner join
```

```
  "sap.hana.democontent.epm.data::MD.Addresses" as b
  on a."ADDRESSES.ADDRESSID" = b.ADDRESSID
  where contains("NAME.LAST", :lastNameFilter, FUZZY(0.7));
END;
```

> Note:
> Notice the *CONTAINS* and *FUZZY* statements in this code. This feature enables you to find the names of employees even if they are misspelled. You will learn more about this in Unit 5.

a) Enter the code for your table function or copy/paste it from the specified file.

b) Save your table function. Alternatively, you can press **Ctrl-S**.

3. Activate your table function.

a) On the top toolbar, choose *Activate* . Alternatively, you can right-click your table function name on the *Repositories* tab and choose *Activate*. You can also press *Ctrl-F3*. Your table function is now ready for use.

Task 3: Use the Table Function in a Dimension Calculation View

1. Create a dimension calculation view.

a) Stay in the *SAP HANA Development* perspective. Right-click your *STUDENT##* package and choose *New → Calculation View*.

b) Specify the name of the calculation view as **CV_FIND_EMPLOYEES**. In the *Data Category*, choose *DIMENSION*. Choose *Finish*.

c) Add your *TF_FUZZY_EMPLOYEES* table function to the *Projection* node.

d) In the *Details* pane, right-click the top bar of the table function field and choose *Add All to Output*.

2. Map the inputs of the table function to an input parameter in your calculation view.

a) Select the *Semantics* node and choose the *Parameters/Variables* tab.

b) In the toolbar, choose the + icon from the drop-down menu and choose *Create Input Parameter*.

c) Enter a name **SURNAME1** and for the *Data Type*, specify **VARCHAR** of length 40. Choose *OK*.

d) In the toolbar, choose *Input Parameter Manage Mappings*.

e) Drag the *LASTNAMEFILTER* name from the left column to the *SURNAME1* name in the right column.
A line should appear connecting these two names.

f) Save your calculation view.
You have just mapped the table function's input to an input parameter. This means that when you preview the calculation view, you will get an input parameter asking you for an employee surname. The value you provide will be passed back to the table function to do a fuzzy text search of the employees with that surname.

3. Use the table function in the calculation view to find employees.

a) Save and activate your calculation view.

b) Preview your calculation view. You will be asked for the value of *SURNAME1*. In the *From* column, type **SMIHT**. This is the surname Smith but typed incorrectly. Choose *OK*.

c) Choose the *Raw Data* tab.
You can see that SAP HANA managed to find two people with the surname Smith, despite you typing the surname incorrectly.

Read Hierarchies via SQL

Shared Hierarchies, defined in Calculation Views of type Dimension and included in the Star Join of a Calculation View of type Cube with Star Join, can be enabled for SQL Access with a flag available in the *Properties* tab of the *Semantics* node. This displays an additional tab in the definition of the hierarchy, where you can specify for example a Node Column name. This column can then be used in a SQL `SELECT` statement, inside the `GROUP BY` clause.

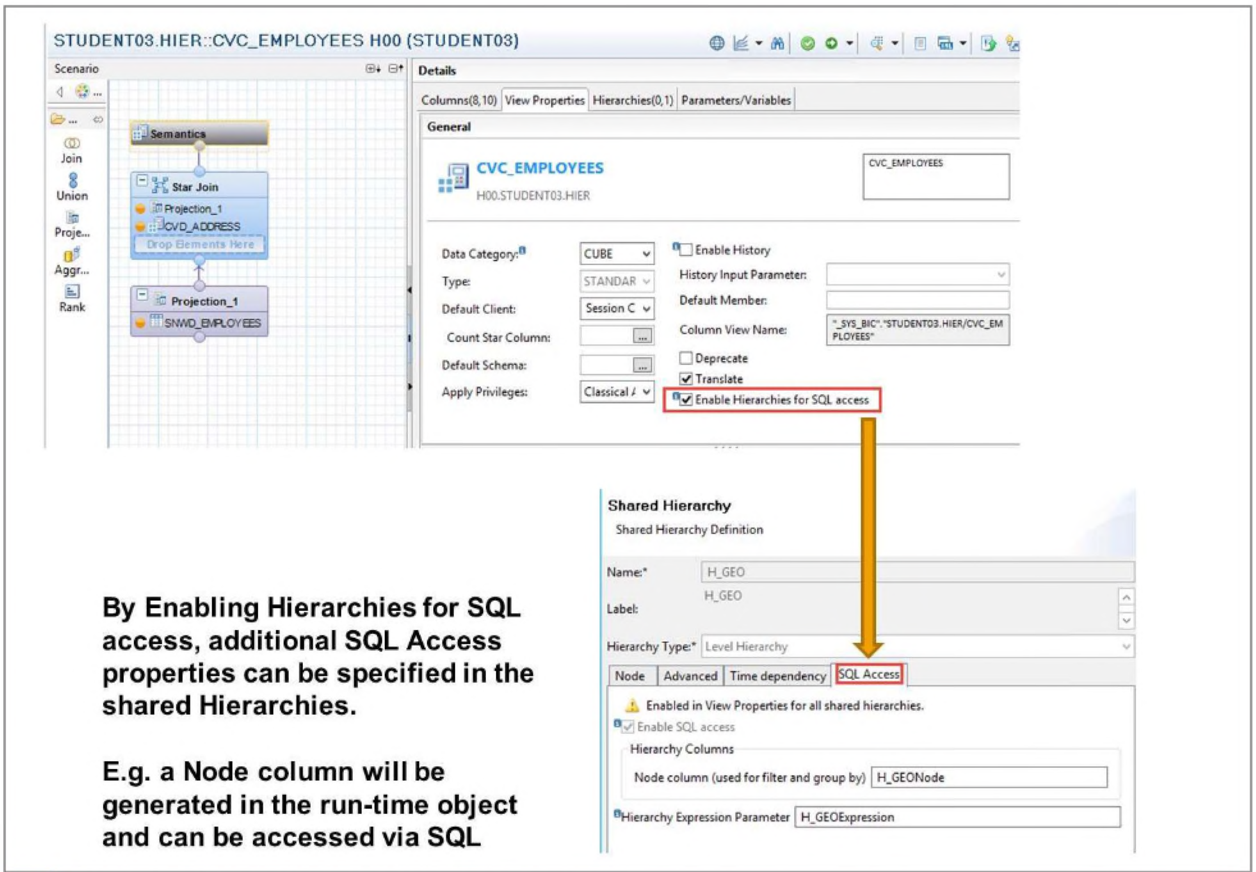


Figure 198: Enabling Hierarchies for SQL Access

Aggregating Values in a Hierarchy via SQL

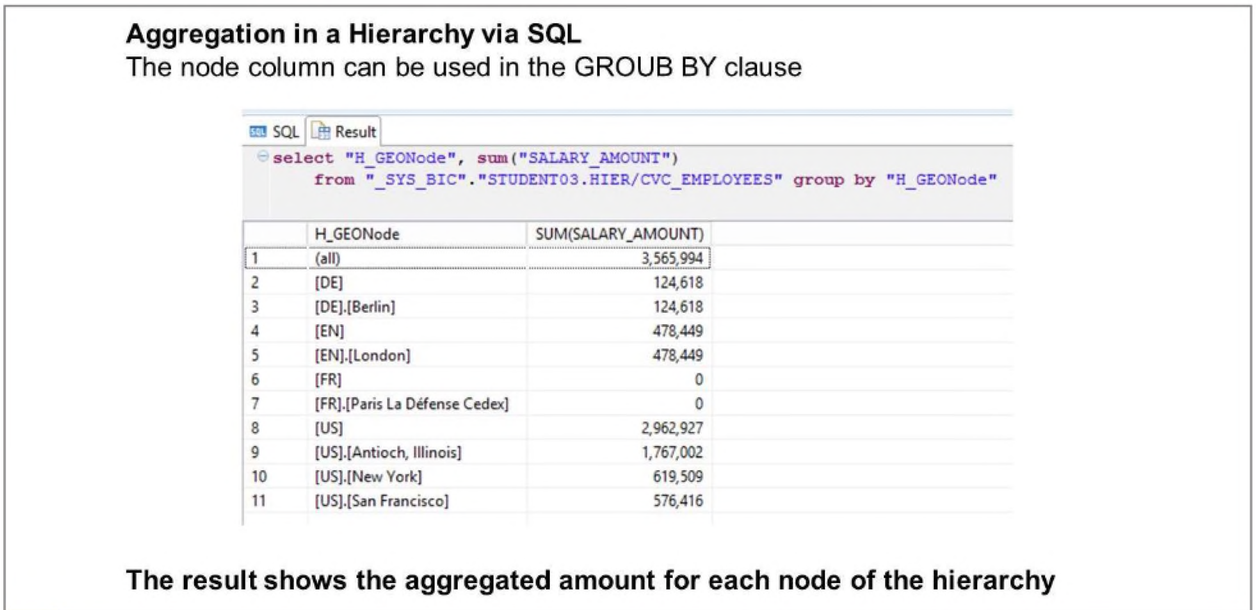


Figure 199: Aggregating Values in a Hierarchy via SQL

Unit 3
Exercise 17

Read a Hierarchy Structure with SQL

Exercise Objectives

After completing this exercise, you will be able to:

- Create a graphical Calculation View of type Dimension
- Create a graphical Calculation View of type Cube with Star Join
- Aggregate values at all levels of a defined hierarchy, with a SQL statement

Business Example

You are using SAP HANA SPS11, and you would like to understand the new features made available since the SPS10 release, These features allow you to query the nodes of a hierarchy with a SQL statement.

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.

Task 1: Build a Dimension Calculation View for Addresses

Create a Calculation View of type Dimension with the employees's addresses, including the city and country, which you will organize into a hierarchy.

1. In your *STUDENT##* package, create a Graphical Calculation View **CVD_ADDRESS_##** (label: **Address**), of type DIMENSION, to expose all fields of the addresses table *EPM_MODEL.SNWD_AD*.

   Define the *CLIENT* and *NODE_KEY* columns as keys.

2. In the *Semantics* node of the *Calculation* View, define a Level Hierarchy **H_GEO** with columns *COUNTRY* and *CITY* as levels. For the Node Style, use *Name Path*.

3. Activate the Dimension Calculation View *CVD_ADDRESS_##*

Task 2: Build a Calculation View for Employees

Create a CUBE with Star Join Calculation View with the Salaries. This view will reuse the Dimension Calculation view *CVD_ADDRESS_##*.

1. In your *STUDENT##* package, create a Graphical Calculation View **CVCS_EMPLOYEES_##** (label: **Employees**), of type *Cube with Star Join*.

2. Create a *Projection* Node to expose the following columns of the employees table *EPM_MODEL.SNWD_EMPLOYEES*. Define this node as the data source for the *Star Join* node.

- CLIENT
- EMPLOYEE_ID
- FIRST_NAME
- LAST_NAME
- SEX
- LANGUAGE
- PR_ADDRESS_GUID
- CURRENCY
- SALARY_AMOUNT

> Hint:
> When adding the columns to the output of the *Projection* node, you can use the *Propagate to Semantics* item from the context menu to include the selected columns up to the output of the *Star Join* node (that is, the *Semantics*) in a single operation.,

3. Add the Dimension Calculation View *CVD_ADDRESS_##* to the *Star Join* node.

   Join the column *PR_ADDRESS_GUID* of the *Projection* node to the column *NODE_KEY* of the Dimension Calculation View. Also join the *CLIENT* columns from both tables.

   Choose a *Referential* join type and an *n..1* cardinality.

4. In the *Semantics* node, check that the column *SALARY_AMOUNT* is defined as a Measure, with the aggregation type *SUM*, and that all other columns are defined as Attributes.

> Note:
> An intermediate “solution” object is available at this stage:
>
> *STUDENT00 → ha300 → CVCS_EMPLOYEES_00_STAGE1*

Task 3: Enable SQL Access to the Hierarchy and Visualize the Data

Eventually, enable the *CVCS_EMPLOYEES_##* Calculation View for SQL access to hierarchies, and display the total salary paid in each city and country.

1. In thevo properties of view *CVCS_EMPLOYEES_##*, enable the hierarchies for SQL access.

2. Activate your Calculation View.

3. Check the total salaries you are paying to your employees, summed at all levels of your geographical hierarchy. Run the following SELECT statement from the SQL Console:

```
select "H_GEONode", sum("SALARY_AMOUNT")
   from "_SYS_BIC"."STUDENT##/CVCS_EMPLOYEES_##"
   group by "H_GEONode";
```

Hint:
You can find this query in your *Student* folder, in the file *HA300 → Read_Hierarchy_Structure_with_SQL.sql*.

	H_GEONode	SUM(SALARY_AMOUNT)
1	(all)	3,565,994
2	[DE]	124,618
3	[DE].[Berlin]	124,618
4	[EN]	478,449
5	[EN].[Lond...	478,449
6	[FR]	0
7	[FR].[Paris ...	0
8	[US]	2,962,927
9	[US].[Anti...	1,767,002
10	[US].[New ...	619,509
11	[US].[San F...	576,416

Figure 201: SQL Query Result

Unit 3
Solution 17

Read a Hierarchy Structure with SQL

Exercise Objectives

After completing this exercise, you will be able to:

- Create a graphical Calculation View of type Dimension
- Create a graphical Calculation View of type Cube with Star Join
- Aggregate values at all levels of a defined hierarchy, with a SQL statement

Business Example

You are using SAP HANA SPS11, and you would like to understand the new features made available since the SPS10 release, These features allow you to query the nodes of a hierarchy with a SQL statement.

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.

Task 1: Build a Dimension Calculation View for Addresses

Create a Calculation View of type Dimension with the employees's addresses, including the city and country, which you will organize into a hierarchy.

1. In your *STUDENT##* package, create a Graphical Calculation View **CVD_ADDRESS_##** (label: **Address**), of type DIMENSION, to expose all fields of the addresses table *EPM_MODEL.SNWD_AD*.

   Define the *CLIENT* and *NODE_KEY* columns as keys.

   a) Right-click your *STUDENT##* package and choose *New* → *Calculation View*. Enter the name **CVD_ADDRESS_##** and label for the view and select the Data Category *DIMENSION*.

   b) Add the table *SNWD_AD* to the *Projection* Node and add all fields to the output.

   c) In the *Semantics* node, select the *Key* checkbox for columns *CLIENT* and *NODE_KEY*.

2. In the *Semantics* node of the *Calculation* View, define a Level Hierarchy **H_GEO** with columns *COUNTRY* and *CITY* as levels. For the Node Style, use *Name Path*.

   a) In the *Hierarchies* tab, create a new Hierarchy with the following properties:

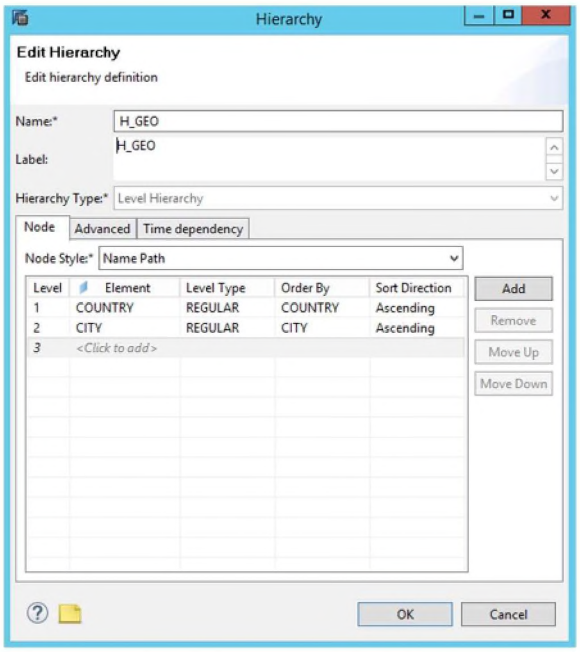


Figure 200: Level Hierarchy with Levels Country and City

b) Choose *OK*.

3. Activate the Dimension Calculation View *CVD_ADDRESS_##*

   a) Choose *Save and Activate*.

   b) Check the outcome in the *Job Log* view.

Task 2: Build a Calculation View for Employees

Create a CUBE with Star Join Calculation View with the Salaries. This view will reuse the Dimension Calculation view *CVD_ADDRESS_##*.

1. In your *STUDENT##* package, create a Graphical Calculation View **`CVCS_EMPLOYEES_##`** (label: **`Employees`**), of type *Cube with Star Join*.

   a) In the *Systems* view, right-click your *STUDENT##* package and choose *NewCalculation* View.

   b) Enter the specified name and description, select the view type *CUBE* and select the *With Star Join* checkbox.

   c) Choose *Finish*.

2. Create a *Projection* Node to expose the following columns of the employees table *EPM_MODEL.SNWD_EMPLOYEES*. Define this node as the data source for the *Star Join* node.

   - CLIENT
   - EMPLOYEE_ID
   - FIRST_NAME
   - LAST_NAME
   - SEX

- LANGUAGE
- PR_ADDRESS_GUID
- CURRENCY
- SALARY_AMOUNT

> Hint:
> When adding the columns to the output of the *Projection* node, you can use the *Propagate to Semantics* item from the context menu to include the selected columns up to the output of the *Star Join* node (that is, the *Semantics*) in a single operation.,

a) From the palette, drag a *Projection* node below the *Star Join* node

b) Draw a connector from the top of the *Projection* node to the bottom of the *Star Join* node.

c) Right-click the *Projection* node and choose *Add Objects*.

d) In the *Find* window, enter the search string **`SNWD_EMP`**, select the *SNWD_EMPLOYEES (EPM_MODEL)* table and click *OK*.

e) Select the specified columns (press the **`Ctrl`** key while clicking them). Then right-click the selection and choose *Propagate to Semantics*.

3. Add the Dimension Calculation View *CVD_ADDRESS_##* to the *Star Join* node.

   Join the column *PR_ADDRESS_GUID* of the *Projection* node to the column *NODE_KEY* of the Dimension Calculation View. Also join the *CLIENT* columns from both tables.

   Choose a *Referential* join type and an *n..1* cardinality.

   a) In the *Scenario* pane, right-click the *Star Join* node and choose *Add Objects*.

   b) In the *Find* window, enter the search string **`CVD_ADD`** and, select the *CVD_ADDRESS (STUDENT##)* table and choose *OK*.

   c) In the *Details* pane, draw two connectors between the specified pairs of columns to define the join.

   d) Double-click any of the two connectors, and define the properties of the join as specified.

4. In the *Semantics* node, check that the column *SALARY_AMOUNT* is defined as a Measure, with the aggregation type *SUM*, and that all other columns are defined as Attributes.

   a) Select the *Semantics* node.

   b) Check the properties of all the Local columns.

> Note:
> An intermediate "solution" object is available at this stage:
>
> *STUDENT00 → ha300 → CVCS_EMPLOYEES_00_STAGE1*

Task 3: Enable SQL Access to the Hierarchy and Visualize the Data

Eventually, enable the *CVCS_EMPLOYEES_##* Calculation View for SQL access to hierarchies, and display the total salary paid in each city and country.

1. In thevo properties of view *CVCS_EMPLOYEES_##*, enable the hierarchies for SQL access.
   a) In the *Semantics* node, select the *View Properties* tab.
   b) Select the *Enable Hierarchies for SQL Access* checkbox. Choose *OK* in the *Confirm* dialog box.
   c) On the *Hierarchies* tab, double-click the (shared) *H_GEO* hierarchy and check the (default) node column name *H_GEONode*.
   d) Keep the default node column and choose *Cancel*.
2. Activate your Calculation View.
   a) Choose *Save and Activate*.
   b) Check the outcome of the activation in the *Job Log* view.
3. Check the total salaries you are paying to your employees, summed at all levels of your geographical hierarchy. Run the following SELECT statement from the SQL Console:

```
select "H_GEONode", sum("SALARY_AMOUNT")
   from "_SYS_BIC"."STUDENT##/CVCS_EMPLOYEES_##"
   group by "H_GEONode";
```

> Hint:
> You can find this query in your *Student* folder, in the file *HA300 → Read_Hierarchy_Structure_with_SQL.sql*.

   a) If you want to copy/paste the SQL statement, open the specified SQL file in Notepad and copy all its content.
   b) In the *Systems* view, select any item from your SAP HANA system tree and choose the *Open SQL Console* icon at the top of the *Systems* view.
   c) Type the SQL statement. Or paste it from the clipboard, and replace any occurrence of **##** with your group number.
   d) Execute the SQL statement.
   e) Analyze the results.

	H_GEONode	SUM(SALARY_AMOUNT)
1	(all)	3,565,994
2	[DE]	124,618
3	[DE].[Berlin]	124,618
4	[EN]	478,449
5	[EN].[Lond...	478,449
6	[FR]	0
7	[FR].[Paris ...	0
8	[US]	2,962,927
9	[US].[Anti...	1,767,002
10	[US].[New ...	619,509
11	[US].[San F...	576,416

Figure 201: SQL Query Result

LESSON SUMMARY

You should now be able to:

- Work with SQLScript
- Explain the SQLScript implementation logic

Unit 3
Lesson 3

Creating and Using Procedures

LESSON OVERVIEW

In this lesson, you will get an introduction to procedures and learn how to define a procedure and how to call the procedure.

Business Example

As part of your SAP HANA implementation project, you have some business requirements for reporting that cannot be fulfilled with only Analytic or Graphical Calculation Views.

You have decided to use SQL Calculation Views, but you would like to get the best possible flexibility by creating procedures that you can easily reuse in several Calculation Views.

You want to know more about procedures and how to use them in the most relevant way.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create and use procedures
- Create a procedure
- Call a procedure

When to Use a Procedure

We have graphical calculation views, user-defined functions (like table functions), and procedures. The question might arise of when you should use each of these?

In order of increasing functionality, we have views, then user-defined functions, and then procedures. Procedures have the most functionality. However, in order of performance we have the reverse. Views give the best performance, then user-defined functions (because they are permanently read-only), and lastly procedures.

We recommend you use the minimal approach to get the results you require. If you can do it with views, then use views. If you cannot do it with views, then try using user-defined functions. If that still does not meet the requirements, create a procedure.

For example, you want to return multiple result sets to a query. You evaluate the options in order from views, to user-defined functions, to procedures.

- Views can only return a single result set.
- So you next have to look at user defined functions.
  - Scalar functions can return multiple single values, but only scalar data types. They cannot return result sets.
  - Table functions can return only a single result set.
- In this case you will finally have to use a procedure.

Introduction to SQLScript Procedures

- Procedures can have multiple input parameters and output parameters (which can be of scalar or table types).
- Procedures describe a sequence of operations on data passed as input and database tables.
- Procedures can in turn call other procedures.
- Read-only procedures can only call other read-only procedures.

Figure 202: Procedures

The use of SQLScript functional-style procedures has some advantages, compared to the use of pure SQL as follows:

- The calculation and transformations described in procedures can be parameterized and reused inside other procedures.
- The user can use and express knowledge about relationships in the data, related computations can share common sub expressions, and related results can be returned using multiple output parameters.
- It is easy to define common sub expressions. The query optimizer decides if a materialization strategy (which avoids re-computation of expressions) or other optimizing rewrites are best to apply. In any case, it eases the task to detect common sub-expressions and improves the readability of the SQLScript code.

SQL Script and Procedures

- Procedure is a reusable processing block. It is implemented using SQL Script.
- A procedure can be created as read only (without side-effect) or read-write.
- Procedure can be implemented using SQL Script, L or R language.

Figure 203: Procedures - Key Properties

The body of a procedure consists of a sequence of statements separated by semicolons.

An intermediate variable, inside a procedure, does not require to be defined before it is bound by an assignment.

Cyclic dependencies that result from the intermediate result assignments or from calling other functions are not allowed.

A variable name is prefixed by a colon (`:`) when it is used as an input to other statement.

Procedures can be created by using the SQL Editor or by using the creation wizard.

> Note:
> In SAP HANA, the L language is used by SAP internally to develop internal functions in AFL libraries.
>
> It is not supported for customers.

Procedures and Analytic Privileges

Analytic Privileges are a type of security artifact used to define data access authorization for Information Views.

A typical use case of procedures is to define Dynamic Analytic Privileges. Analytic Privileges in which the data access restrictions are not fixed, but evaluated on the fly –in particular, based on the user– when an Information View is queried.

> Note:
> For example, a procedure can return for a specific user the list of countries for which this user can access the sales data. It can be used to create a single Analytic Privilege that will secure dynamically a *Sales* Calculation View for all the users.

Creating a Procedure

Syntax:

```
CREATE PROCEDURE {schema.}name
  {({IN|OUT|INOUT}
              param_name data_type {,...})}
    {LANGUAGE <LANG>} {SQL SECURITY <MODE>}
    {READS SQL DATA {WITH RESULT VIEW <view_name>}} AS
BEGIN
 ...
END
```

READS SQL DATA defines a procedure as read-only.

Implementation **LANGUAGE** can be specified . Default is SQLScript.

WITH RESULT VIEW is used to create a column view for output parameter of type table which can be used in SQL query.

Figure 204: SQLScript: Procedure Creation using SQL Editor - SQLScript and Procedures

- Language

  The implementation language is by default SQLSCRIPT. It is a best practice to define the language in every procedure definition. Other implementation languages, such as the *R language* and *L language*, are supported, but not covered in this introductory unit.

- Security Mode

With security mode `Definer's rights (user _SYS_REPO)`, which is the default, the execution of the procedure is then performed with the privileges of user `SYS REPO`. On activation, the technical user `_SYS_REPO` creates the runtime procedure (in the `_SYS_BIC` schema) and is then considered as the definer of the procedure.

The other alternative is mode `Invoker`. In this case, privileges are checked at runtime with the privileges of the caller of the function.

> Note:
> Analytic privileges are always checked, regardless of the security mode.

- READS SQL DATA

  `READS SQL DATA` is used to define a procedure as read-only and marks a procedure as being free of side-effects. Neither DDL nor DML statements are allowed in its body, and only other read-only procedures can be called by the procedure. The advantage to this definition is that certain optimizations are only available for read-only procedures.

- WITH RESULT VIEW

  `RESULT VIEW` can be specified for read-only procedures (those identified by the `READS SQL DATA` keyword). It is used to specify a view that is used as the output of the procedure. If a result view is specified for a procedure, the procedure can be called by an SQL statement in the same way as a table or a view. The name of the result view is no longer bound to a static name scheme but can be any valid SQL identifier. For backward compatibility reasons the old static name scheme will be only supported for procedures that are generated with the deprecated `CREATE FUNCTION` syntax.

Input and Output parameters are not mandatory but in most cases you will define some to ensure your procedures are modular and allow better reuse.

Calling a Procedure

CALL - Procedure Called From Client
A procedure (or table function) can be called by a client on the outer-most level, using any of the supported client interfaces.

Syntax
CALL [schema.]name (param1 [, ...])

SQL | Result

```
call "_SYS_BIC"."student03/MICHAEL_PROC" (input_int => 3, output_int => ?);
```

For table output parameters it is possible to either pass a table or '?'.
'?' can be used to represent an empty parameter binding.

Figure 205: Calling a Procedure

When calling a procedure using the `CALL` statement, it behaves in a consistent way as the SQL standard semantics, for example, Java clients can call a procedure using a JDBC Callable statement. `CALL` returns an iterator over result sets. Each output variable of the procedure is represented as a result set.

CALL WITH OVERVIEW

CALL...WITH OVERVIEW From Client

This CALL statement returns one result set that holds the information of which table contains the result of a particular table's output variable.
This is used to populate an existing table by passing it as parameter.
When passing '?' to the output parameters, temporary tables holding the result sets will be generated.

Syntax
CALL [schema.]name (param1 [, ...]) WITH OVERVIEW

SQL | Result

```
call "_SYS_BIC"."student03/MICHAEL_PROC"
(input_int => 3, output_int => ?)
WITH OVERVIEW;
```

	variable	table
1	OUTPUT_INT	"STUDENT03"."OUTPUT_INT_50AB74A47460572CE10000000A1671F8"

Figure 206: Calling a Procedure with Overview

This `CALL` statement returns one result set that holds the information of the table that contains the result of a particular table's output variable.

When passing existing tables to the output parameters, the `CALL WITH OVERVIEW` inserts the result set tuples of the procedure into the given tables.

Calling a Procedure from a Script-Based Calculation View

CALL – Internal Procedure Call

For internal calls, i.e. calls to a procedure by a calculation view or another procedure, IN variables are bound by literals or variable references, new OUT variables are bound to the result of the call.

Syntax
CALL [schema.]name (:in_param1, out_param [, ...])

```
BEGIN

it_pubs =   SELECT PUB_ID AS PUBLISHER, NAME
            FROM PUBLISHERS;

CALL "_SYS_BIC"."STUDENT00/P_BOOKS_VAT_00" (:INP_TAX_RATE, it_books);

var_out =   SELECT P.PUBLISHER, P.NAME, B.ISBN, B.TITLE, B.PRICE, B.CURRENCY, B.PRICE_VAT
            FROM :it_pubs P, :it_books B
            WHERE P.PUBLISHER=B.PUBLISHER;
```

Figure 207: Calling a Procedure

Procedures can be called with this syntax by another procedure or a calculation view. In the case of a script-based calculation view, it is possible to pass the IN parameters to the procedure by hard-coding the values of these parameters in the procedure call, or by requesting these parameters to the end-user like in any information view with input parameters.

> Note:
> In the figure, Calling a Procedure, `:INP_TAX_RATE` refers to an input parameter defined in the calculation view.

Activating Design-Time Procedures and Table Types

Upon activation, the procedures created with a design-time object are owned by the technical user `_SYS_REPO`. They are located as follows:

Where to Find Activated Procedures?

- Procedures created in the Modeler Perspective

  These procedures are located in the `_SYS_BIC` schema, under *Procedures*.

- Procedures created with a CDS document, with extension .hdbprocedure

  These procedures are located in the schema specified in the CDS document.

> Note:
> The procedures created by an SQL Statement, without using a design-time object, are owned by the user that executes the `CREATE PROCEDURE` statement.
>
> They are located in the schema specified by the statement.

Specifying the Table Types for IN and OUT parameters

When a procedure uses a tabular input or a tabular output, the corresponding table type(s) must be created to successfully activate the procedure. There are two options to define the table types:

- Define externally a global table type, for example, by using Core Data Services (CDS) syntax.

  The activation of the table type is done separately from the activation of the procedure(s) that refer to it.

- Define the table type inline, for example, in the procedure itself.

  The activation of the table type is done along with the procedure.

Unit 3
Exercise 18

Create a Procedure

Exercise Objectives

After completing this exercise, you will be able to:

- Create a procedure

Business Example

You need to add a tax calculation to the basis price for your books sales. To enable flexible input of different tax rates, you decide to create a procedure.

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.

Task 1: Create a New Procedure

1. Open the *SAP HANA Development* perspective and create a local repository workspace for system *H00 (STUDENT##)*.

2. Create a new procedure **P_BOOKS_VAT_##** of type *HDBProcedure* in your package *STUDENT##*. This procedure must be defined with target schema *STUDENT##*.

3. Create the code of your procedure.

   You now need to type or copy the following SQL Script code in your procedure. Make sure you replace ## with your group number everywhere in this code.

   Note that the procedure includes a scalar input parameter (tax) and an output parameter of type table (out_with_tax) that is defined inline, that is, within the code of the procedure itself.

```
PROCEDURE "STUDENT##"."STUDENT##::P_BOOKS_VAT_##"
(IN tax DECIMAL(5,2),
 OUT out_with_tax table
  (ISBN VARCHAR (20),
   TITLE VARCHAR (50),
   PUBLISHER INT,
   PRICE DECIMAL (33,2),
   PRICE_VAT DECIMAL (33,2),
   CURRENCY VARCHAR (3)
  )
)

LANGUAGE SQLSCRIPT
SQL SECURITY DEFINER
DEFAULT SCHEMA STUDENT##
READS SQL DATA AS
```

```
/********* Begin Procedure Script ************/
BEGIN

out_with_tax = SELECT ISBN,
                      TITLE,
                      PUBLISHER,
                      PRICE,
                      PRICE * :tax AS PRICE_VAT,
                      CRCY AS CURRENCY
               FROM BOOKS;

END;
/********* End Procedure Script ************/
```

> Note:
> The SQLScript code can be copied from a text file in your *student* folder: *ha300 → P_BOOKS_VAT_TEMPLATE.txt*.

4. Activate the procedure and check the activation log.

5. In the *Systems* view, check that the activated procedure *STUDENT##::P_BOOKS_VAT_##* is available in the schema *STUDENT##*, in the *Procedures* folder.

> Hint:
> To see the procedure, you must select the *Show All Objects* checkbox in the preferences of SAP HANA Studio. This option is located in *SAP HANA → Modeler → Content Presentation*.

6. Back to the *Repositories* view, reset your local repository workspace.

Task 2: Test your Procedure and Review the Results

1. Return to the *SAP HANA Modeler* perspective, and open a new SQL Editor tab.

   Write the following SQL statement to call your procedure with a *tax* input parameter value of 1.19:

   ```
   call "STUDENT##"."STUDENT##::P_BOOKS_VAT_##" (1.19, ?);
   ```

   Execute your statement and review the *Result* tab (you can try different tax rates for the calculation).

2. Using the tax rate 1.25, fill the table below with the price including VAT:

Title	Price Including VAT
Datenbanksysteme: Eine Einfuehrung	
Grundlagen von Datenbanken	

Create a Procedure

Exercise Objectives

After completing this exercise, you will be able to:

- Create a procedure

Business Example

You need to add a tax calculation to the basis price for your books sales. To enable flexible input of different tax rates, you decide to create a procedure.

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.

Task 1: Create a New Procedure

1. Open the *SAP HANA Development* perspective and create a local repository workspace for system *H00 (STUDENT##)*.

   a) In the top right of the SAP HANA studio, choose the *SAP HANA Development* perspective.

   b) In the *Repositories* view, right-click your *H00 (STUDENT##)* system and choose *Create Repository Workspace*.

   c) In the *Create New Repository Workspace* dialog, keep the default values and choose *Finish*.
   When you expand the workspace, you will see a list of the packages in your H00 system.

2. Create a new procedure **`P_BOOKS_VAT_##`** of type *HDBProcedure* in your package *STUDENT##*. This procedure must be defined with target schema *STUDENT##*.

   a) Right-click your *STUDENT##* package and choose *New → Other... → SAP HANA → Database Development → Stored Procedure*.

   b) Choose *Next*.

   c) In the *File name* field, enter **`P_BOOKS_VAT_##`**.

   d) Next to the *Target Schema* field, click the *Browse button*, select your *STUDENT##* schema and choose *OK*.

   e) Choose *Finish*.

3. Create the code of your procedure.

You now need to type or copy the following SQL Script code in your procedure. Make sure you replace ## with your group number everywhere in this code.

Note that the procedure includes a scalar input parameter (tax) and an output parameter of type table (out_with_tax) that is defined inline, that is, within the code of the procedure itself.

```
PROCEDURE "STUDENT##"."STUDENT##::P_BOOKS_VAT_##"
(IN tax DECIMAL(5,2),
 OUT out_with_tax table
   (ISBN VARCHAR (20),
    TITLE VARCHAR (50),
    PUBLISHER INT,
    PRICE DECIMAL (33,2),
    PRICE_VAT DECIMAL (33,2),
    CURRENCY VARCHAR (3)
   )
)

LANGUAGE SQLSCRIPT
SQL SECURITY DEFINER
DEFAULT SCHEMA STUDENT##
READS SQL DATA AS

/********* Begin Procedure Script ************/
BEGIN

out_with_tax = SELECT ISBN,
                      TITLE,
                      PUBLISHER,
                      PRICE,
                      PRICE * :tax AS PRICE_VAT,
                      CRCY AS CURRENCY
               FROM BOOKS;

END;
/********* End Procedure Script ************/
```

> Note:
> The SQLScript code can be copied from a text file in your *student* folder: *ha300 → P_BOOKS_VAT_TEMPLATE.txt*.

a) From Windows Explorer, in your *student* folder, double-click the file *ha300 → P_BOOKS_VAT_TEMPLATE.txt*.
The file opens in Notepad.

b) Select all the code and press Ctrl+C.

c) In the SQLScript editor of your new procedure, remove all existing content and, to paste the code, press Shift+Insert (or right-click the blank space and choose *Paste*).

4. Activate the procedure and check the activation log.

a) In the *Repositories* view, right-click your *P_BOOKS_VAT_##* procedure and choose *Activate*.

b) Alternatively, you can click the *Activate SAP HANA Development Object* button from the toolbar.

c) In the *Job Log* view, check the job status is *Completed Successfully*.

5. In the *Systems* view, check that the activated procedure *STUDENT##::P_BOOKS_VAT_##* is available in the schema *STUDENT##*, in the *Procedures* folder.

> Hint:
> To see the procedure, you must select the *Show All Objects* checkbox in the preferences of SAP HANA Studio. This option is located in *SAP HANA → Modeler → Content Presentation*.

   a) In the Content of the *Systems* view, expand the folder *STUDENT## → HDBProcedures*.

   b) Check that the runtime procedure has been created in the folder.

   c) In the SAP HANA Studio preferences, deselect the *Show all objects* checkbox. to come back to the default content presentation.

6. Back to the *Repositories* view, reset your local repository workspace.

   a) In the *Repositories* view, right-click the *Imported Workspace (default)* object just below the *H00 (STUDENT##)* system and choose *Reset*.

   b) In the *Reset Repository Workspace* dialog box, select the reset mode *Reset* and choose *OK*.

Task 2: Test your Procedure and Review the Results

1. Return to the *SAP HANA Modeler* perspective, and open a new SQL Editor tab.

   Write the following SQL statement to call your procedure with a *tax* input parameter value of 1.19:

```
call "STUDENT##"."STUDENT##::P_BOOKS_VAT_##" (1.19, ?);
```

   Execute your statement and review the *Result* tab (you can try different tax rates for the calculation).

   a) In the top right of the SAP HANA studio, choose the *SAP HANA Modeler* perspective.

   b) In the *Systems* view, right-click the SAP HANA instance and choose *Open SQL Console*.

   c) Type the SQLScript code in the SQL Editor

   d) To execute your statement, press F8.

   e) Check the output in the *Result* tab.

2. Using the tax rate 1.25, fill the table below with the price including VAT:

Title	Price Including VAT
Datenbanksysteme: Eine Einfuehrung	
Grundlagen von Datenbanken	

   a) Modify the SQL statement to use a tax rate of 1.25:

```
call "STUDENT##"."STUDENT##::P_BOOKS_VAT_##" (1.25, ?);
```

   b) Check the results for the two specified books.

Title	Price Including VAT
Datenbanksysteme: Eine Einfuehrung	49.75
Grundlagen von Datenbanken	37.43

LESSON SUMMARY

You should now be able to:

- Create and use procedures
- Create a procedure
- Call a procedure

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

Unit 3
Lesson 4

Using SQLScript Debug Tools

LESSON OVERVIEW

In this lesson, you will get an introduction to the tools to use to debug SQLScript procedures, focusing on SAP HANA Studio Capabilities.

Business Example

As a modeler, you are interested in implementing procedures in your SAP HANA system to make your information models more dynamic, but you need a tool to help you solve the issues that arise when you are testing these artifacts.

You want to learn more about the tools offered in the SAP HANA Studio environment.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Use SQLScript Debug Tools

SQLScript Debug Tools

SAP HANA Provides tools to debug SQLScript procedures in SAP HANA Studio, and in the Web-based Development Workbench.

In SAP HANA Studio, the tool provides the following capabilities:

Main Features of the Debug Tool in SAP HANA Studio

- Debug both catalog and repository procedures
- Set breakpoints in the procedure code to debug step-by-step
- Display the value of scalar and table variables at any breakpoint
- Debug sub-procedures called by the main one on-the-fly
- Store several debug configurations for the same procedure and apply them during a new debug session

Debug Perspective

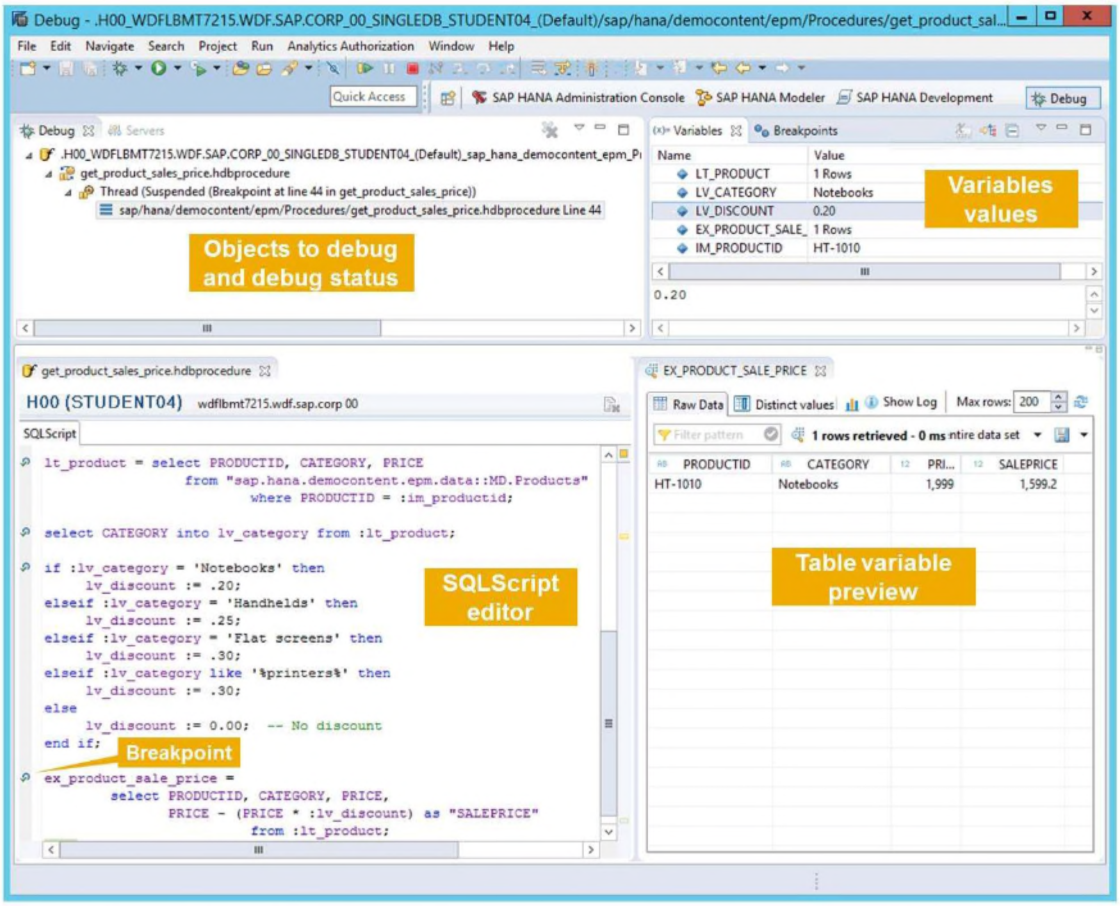


Figure 208: The Debug Perspective

The *Debug* perspective includes a number of views to facilitate debugging.

- The SQLScript editor view is where you set breakpoints and monitor their status after execution. It allows you to edit, then save, the script of your procedure.
- The Debug view list the procedures opened in the debug session, their status (suspended, terminated, and so on.)

  You can suspend, resume, terminate and relaunch the debug thread.
- The Variables view displays, at each breakpoint, the value of the scalar variables and the number of rows of variables of type table.

  It is possible to display in a dedicated preview the content of variables of type table.
- The Breakpoints view lists all the breakpoints you created, and allows you to select/deselect them before a debug execution.

To Debug a Procedure

Your user must have the relevant authorizations to debug the procedure.

1. Open the procedure from the *Repositories* view of the *SAP HANA Development* perspective.

   The location of the object to open depends on the type of procedure:

- For a repository procedure, open the design-time object from its containing package.

> Note:
> To open a repository procedure, the object must be imported or copied into your local workspace.
>
> If needed, the local workspace can be created on the fly. Confirm the workspace location or modify it.

- For a catalog procedure, open the (runtime) object from its containing schema.

2. Add relevant breakpoints to the SQLScript code, in particular, one just before the `END;` terminal, to show the value of the `OUT` variables.

> Hint:
> To add (or remove) a breakpoint, double-click the light-blue vertical bar on the left of the SQLScript editor.

3. Switch to the *Debug* perspective, and in the SQL Editor that contains the procedure script, right-click and choose *Debug as → SAP HANA Stored Procedure*.
4. Fill in the debug settings (in particular the value of input parameters) and choose *Debug*.
5. Check the values of the variables at each breakpoint, and press **F8** to resume the debug session and proceed with the next portion of script.
6. To finish your debug session, in the *Debug* view, right-click the procedure and choose *Terminate and Remove*
7. Close the procedure in the SQLScript Editor (click the X icon).
8. Close the *Debug* perspective.

Required Authorizations to Debug a Procedure

To be able to debug a procedure, a user must have the following privileges:

- Object Privilege *EXECUTE* on the *DEBUG (SYS)* procedure.
- Object Privilege *DEBUG* on the procedure, or on the schema containing it.

Note:
The schema containing a procedure is determined as follows:

- Catalog procedures are located in the schema in which the procedure has been created.
- Repository procedures with design-time object of type *.procedure* (for example, the procedures created in the Modeler perspective) have their activated version in the schema *_SYS_BIC*
- For repository procedures with design-time object of type *.hdbprocedure* (for example, CDS entities of type procedure, created in the Developer perspective), the design-time object defines the SCHEMA in which the procedure must be created upon activation.

Debugging for Another User's Session

It is possible to debug a procedure for another user.

In this case, the debugger needs to be granted the *ATTACH DEBUGGER* privilege by the owner of the procedure. This is a specific type of privilege: *Privilege on User*.

Caution:
During a debug session, the external debugger can see the same data as the owner of the procedure, especially without being granted additional System Privileges or Analytic Privileges. So granting the *ATTACH DEBUGGER* privilege must be done carefully.

LESSON SUMMARY

You should now be able to:

- Use SQLScript Debug Tools

Understanding Deprecated SQL Modeling Objects

LESSON OVERVIEW

This lesson describes the recent evolution of the modeling approach for scripted Information Views, to explain the main reasons for this evolution, and also give you some knowledge about the deprecated scripted artifacts, because you might see them in previous implementations of SAP HANA.

Business Example

You work on an SAP HANA project for a customer who has decided to upgrade their SAP HANA Systems to SAP HANA SPS11.

You are in charge of analyzing the existing scripted information models, and make recommendations on how to adopt the most up-to-date modeling approaches.

The purpose is to ensure that the new models perform better, they are easy to maintain, and also make the transition to the new approach as smooth as possible.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Understand Deprecated Modeling Objects

Scripted Calculation Views - Former Approach

In the previous versions of SAP HANA, script-based calculation views were mainly used for the following purposes:

- As an alternative to Graphical Information Views.

  In some scenarios, you had to perform complex computation of data that the graphical models could not address.

- To force the execution of some very heavy data computations in the Calculation Engine.

  In some scenarios, due to the complexity of the data computation and/or the very high volume of data to process, you had to by-pass the SQL Optimizer, which in some scenarios did not provide the best optimization.

  A particular type of SQL Script operators, called Calculation Engine Plan Operators (also named CE Functions) was used for this purpose. This was a way to achieve better performance than what you could expect with graphical models or with scripted models using only pure SQL.

Catching Up with the Latest Versions of SAP HANA

We have seen in the exercise how to use a table function in a graphical calculation view. This way of working, which was not possible before, changes the way we approach modeling in

SAP HANA projects. The emphasis now has moved completely to the graphical modeling environment.

We can work graphically with our SAP HANA views, create user-defined functions or procedures for the occasional constraint, and bring them back into the graphical SAP HANA modeling environment. We can then continue building the remainder of our SAP HANA information models using the graphical information views.

Many of the SAP HANA modeling constructs and objects in SQLScript have changed to accommodate this shift in emphasis to the graphical environment. In the last few years almost everything has changed!

Let us contrast the old with the new.

Older versions of SAP HANA	In the latest versions of SAP HANA
Create run-time objects • Example: `CREATE PROCEDURE`	Create design-time objects • Example: Create a .hdbprocedure Run-time object
In the SQL Console Use SQLScript "CE_" functions • Example: `CE_UNION_ALL(:A, :B)`	Use SQL syntax • Example: `SELECT * FROM :A` `UNION ALL` `SELECT * FROM :B`
SQLScript (scripted) calculation views Create a procedure in the Content area OR Create a .procedure file `CREATE TABLE TYPE`	Table functions (.hdbtablefunction) Create a .hdbprocedure file `CREATE TYPE`

Figure 209: Changes in SQL and SQLScript

The way we use SQL and SQLScript in the latest versions of SAP HANA can be summarized as follows:

- You create these in the SAP HANA Development perspective.

  Don't create scripted calculation views in the modeling perspective or create procedures in the SQL Console.
- Create design-time objects like CDS views, `.hdbprocedure`, and `.hdbtablefunction`.
- The CE-functions are no longer used.
- Create the minimum in SQL or SQLScript, then bring them back into the graphical modeling environment and use them there.
- You can migrate old scripted calculation views using the Migration Wizard.

LESSON SUMMARY

You should now be able to:

- Understand Deprecated Modeling Objects

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

Unit 3

Learning Assessment

1. SAP HANA SQLScript allows pushing data intensive logic into the database.

   Determine whether this statement is true or false.

   - [ ] True
   - [ ] False

2. When activating a design-time procedure, the corresponding runtime object is owned by user `_SYS_REPO`.

   Determine whether this statement is true or false.

   - [ ] True
   - [ ] False

3. In a procedure, what are the possible security modes?

   Choose the correct answers.

   - [ ] A Definer
   - [ ] B Analytic Privileges
   - [ ] C Invoker
   - [ ] D Application Privilege

4. Which of the following scripted modeling artifacts is recommended to achieve better flexibility and reusability?

   Choose the correct answer.

   - [ ] A Table Functions
   - [ ] B Scripted Calculation Views

Learning Assessment - Answers

1. SAP HANA SQLScript allows pushing data intensive logic into the database.

   Determine whether this statement is true or false.

   [X] True

   [ ] False

2. When activating a design-time procedure, the corresponding runtime object is owned by user `_SYS_REPO`.

   Determine whether this statement is true or false.

   [X] True

   [ ] False

3. In a procedure, what are the possible security modes?

   Choose the correct answers.

   [X] A Definer

   [ ] B Analytic Privileges

   [X] C Invoker

   [ ] D Application Privilege

4. Which of the following scripted modeling artifacts is recommended to achieve better flexibility and reusability?

   Choose the correct answer.

   [X] A Table Functions

   [ ] B Scripted Calculation Views

UNIT 4 Virtual Data Models in SAP HANA

UNIT OBJECTIVES

- Get started with SAP HANA Live
- Understand the Virtual Data Model
- Discover and consume SAP HANA Live views
- Modify SAP HANA Live Views
- Use the SAP HANA Live Extension Assistant
- Leverage CDS Views to create a Virtual Data Model

Unit 4
Lesson 1

Getting Started with SAP HANA Live

LESSON OVERVIEW

SAP HANA Live is an optional extension to SAP HANA and its purpose is to provide ready to use business views based on SAP Business Suite tables. In this lesson you will discover why implementing SAP HANA Live is essential in the provision of live operational information to business reports and applications.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Get started with SAP HANA Live

SAP HANA Live

SAP HANA Live for SAP Business Suite is a comprehensive set of predefined SAP HANA models based on SAP Business Suite tables.

Due to the normalization of the database schema found in SAP Business Suite applications, tables can appear complex, fragmented, and are often difficult to consume without a thorough understanding of each schema. SAP HANA Live provides ready to use business views over the tables so that report developers and application builders can easily consume business information in real time without being concerned about the underlying tables. The views are created by SAP but can be modified by customers.

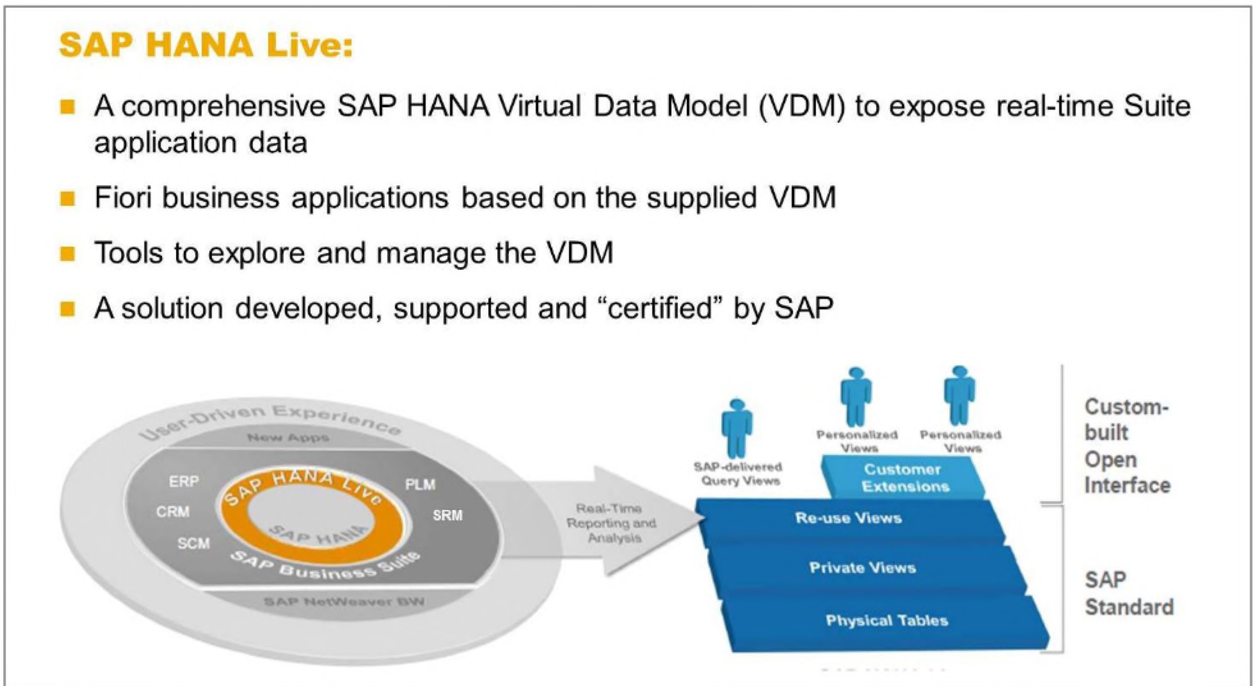


Figure 210: What is SAP HANA Live?

SAP HANA Live models contain all of the necessary joins, filters, aggregations, and transformations to turn the data in the raw suite tables into meaningful information.

SAP HANA Live models are compiled into column views that are technically no different from the column views created from custom modeling efforts.

SAP HANA Live views can be consumed by any SAP Business Objects reporting tool, or by any custom development where a column view is required.

SAP HANA Live views are a key component in SAP S/4HANA and are required to power many of the new S/4HANA applications such as Simple Finance. So a solid understanding of SAP HANA Live is essential for any application and technical consultant working in an SAP S/4HANA project.

Most SAP HANA Live tools are integrated into SAP HANA studio, however, some SAP HANA Live tools are standalone.

SAP HANA Live has its own release cycle, so check the versions available and to evaluate which version will work with your suite and SAP HANA solutions.

What SAP HANA is not

- SAP HANA Live is not a Rapid Deployment Solution (see notes ** below)
- SAP HANA Live is not an Accelerator
- SAP HANA Live does not make SAP BW obsolete
- With SAP HANA Live focuses on live operational reporting on Business Suite data. It does not address Enterprise Data Warehouse use cases

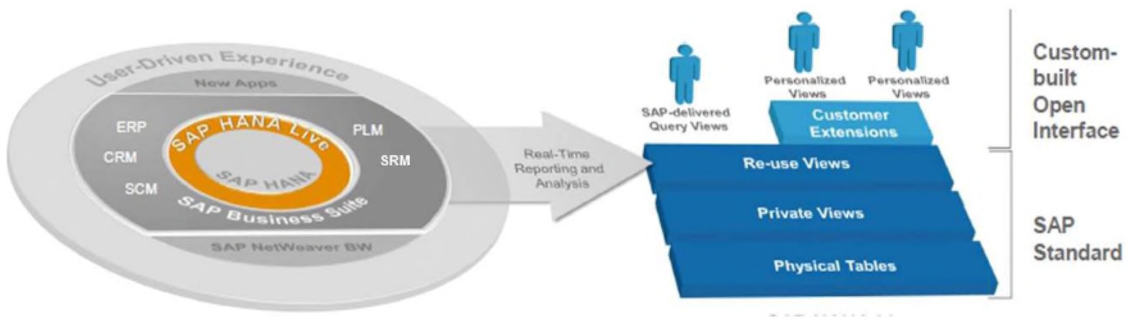


Figure 211: SAP HANA Live - What it is not

- There is a separate Rapid Deployment Solution (RDS) for SAP HANA Live, however, do not confuse SAP HANA Live with the RDS for SAP HANA Live.
- SAP HANA Live provides models and model exploration tools. SAP HANA Live RDS provides SAP BI reporting content based on SAP HANA Live models.
- SAP HANA Live does not require SAP HANA Live RDS.
- SAP HANA Live RDS does require SAP HANA Live.

Comprehensive Suite Coverage

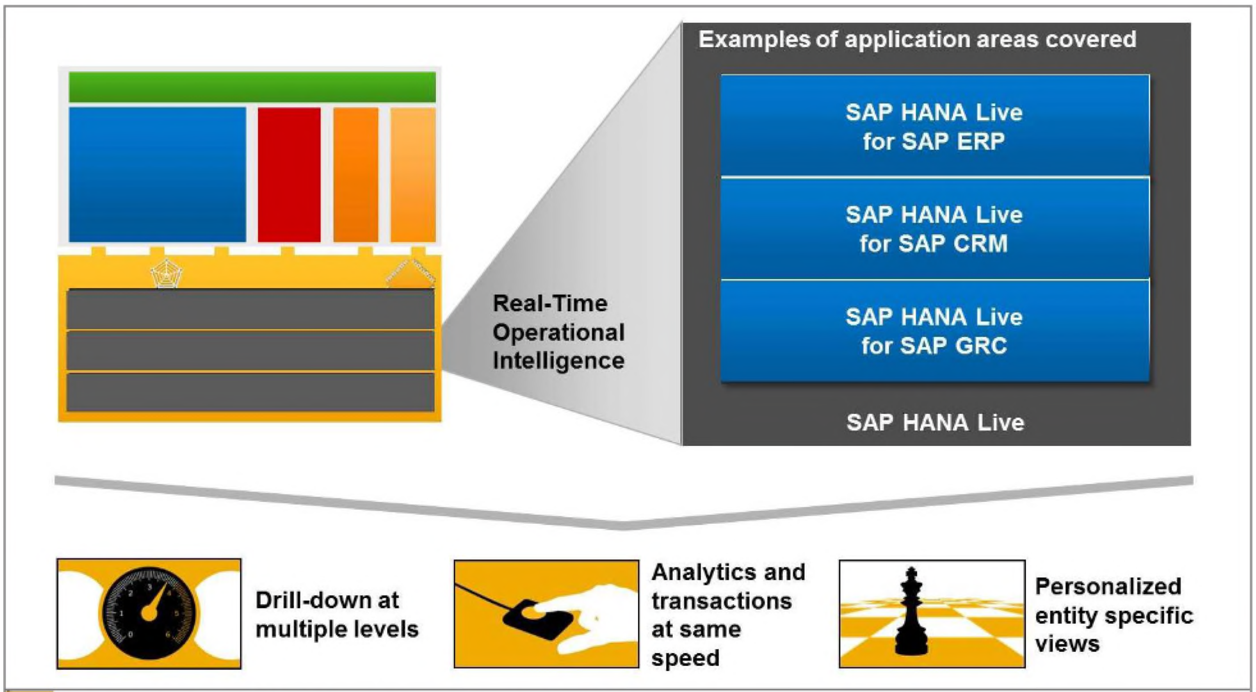


Figure 212: SAP HANA Live - Comprehensive Suite Coverage

SAP HANA Live is delivered in SAP HANA packages covering a large number of suite application and specific solution areas (ERP, GRC, and CRM).

You install only the SAP HANA Live packages you need from Service Marketplace.

SAP HANA Live Combined with SAP Business Warehouse (BW)

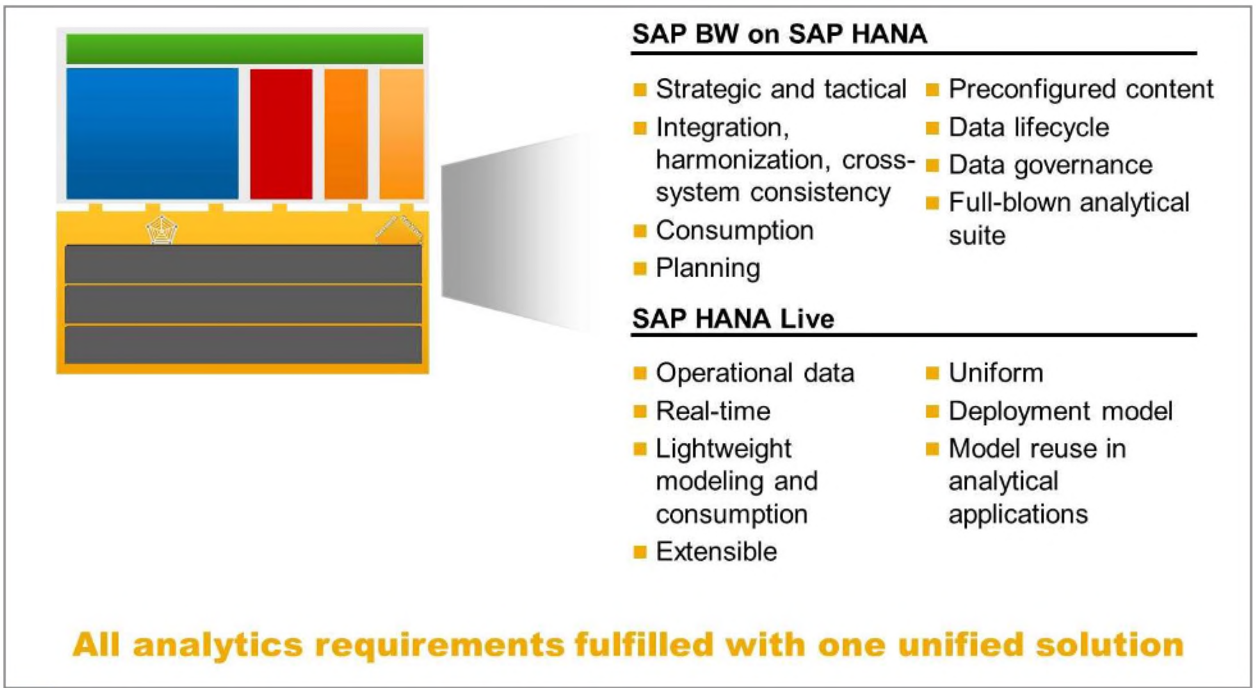


Figure 213: SAP HANA Live Combined with SAP BW

SAP HANA Live combined with SAP Business Warehouse (BW) provides the best-of-breed suite of analytical services (especially when SAP BW is installed on SAP HANA).

Although there are overlaps with SAP HANA Live, and a data mart could easily be developed with SAP HANA, SAP BW is the recommended solution for scalable Enterprise Data Warehouse (EDW) scenarios. BW has many features that are not found in core SAP HANA or SAP HANA Live. These features are as follows.

- Disciplined architecture model (LSA)
- ETL with full referential integrity and powerful delta management built in
- Automated data loading controls (process chains) and error handling
- Built in planning applications
- Data governance controls (data flow templates)
- Fully integrated Lifecycle Management (NLS)
- ABAP transformations (to develop complex business level logic)
- Sophisticated OLAP security modeling
- Modeled (external) hierarchies
- Advanced OLAP features

Two Deployment Options for SAP HANA Live

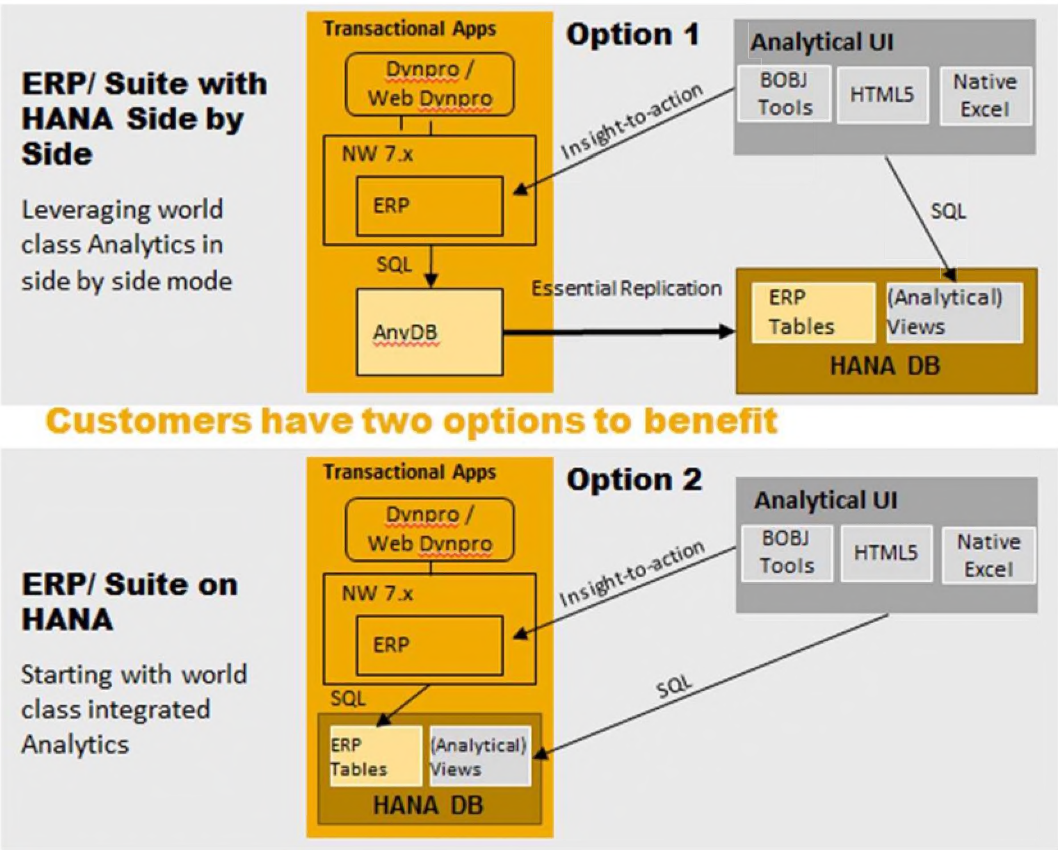


Figure 214: Two Deployment Options for SAP HANA Live

A key decision that customers must make is to decide whether to implement SAP HANA Live as an integrated scenario or a side-by-side deployment.

Side-by-side deployment requires the implementation of a data loading tool to move data from the tables in the source suite application to SAP HANA. SAP Landscape Transformation

Replication Server (SLT) is recommended, but other data provisioning tools can be used. The key objective is to ensure operational tables are replicated to SAP HANA. One of the most powerful tools used to explore SAP HANA Live models is the SAP HANA Live Browser. The SAP HANA Live Browser is able to generate a list of tables required to support any SAP HANA Live model.

One of the major benefits of using SLT for table replication is that it can consume the list of tables produced by SAP HANA Live Browser. This makes setting up replication much easier. The alternative is to enter all tables manually. Also please remember, with a side car deployment, the data exposed by the SAP HANA Live view is only as up-to-date as the last replication of data from the source tables. So if you set you replication to 10 minute intervals you really do not have a live view of your source data.

Ideally, all operational data should be replicated in real time, hence the name, SAP HANA Live.

With the integrated deployment scenario, SAP HANA Live is installed inside SAP HANA that is powering the suite application. This means the tables used by the suite application are also now shared by the SAP HANA Live views. No data movement takes place, there is zero redundancy and this means that the SAP HANA Live views always reflect real-time information.

> Note:
> Whichever deployment scenario you choose, you must define the schema mapping before you import the packages. During the package import the models will have their schema name automatically swapped from the schema name supplied by SAP to the schema name where your suite tables are found in your system. The schema mapping table is used to identify the target schema name that is matched to each SAP authoring schema. Failure to take care of the schema mapping will mean SAP HANA Live views retain their original SAP authoring schema, and therefore they will not activate because the schema the SAP HANA Live views refer to will not be present in your system.

Benefits of Implementing SAP HANA Live

The following are benefits of implementing SAP HANA Live:

- Integrated deployment, reduction of TCO by combining OLTP and OLAP. You no longer need separate transactional systems and operational reporting systems.
- Improved quality of analytics with real time data availability. Make decisions based on up to date, trusted information.
- Avoid having to recreate SAP HANA models for SAP Suite applications thereby improving speed to deployment.

LESSON SUMMARY

You should now be able to:

- Get started with SAP HANA Live

Unit 4
Lesson 2

Understanding the Virtual Data Model

LESSON OVERVIEW

Understanding the architecture of SAP HANA Live is essential to maximize its value. In this lesson, you will learn, in detail, about the architecture of SAP HANA Live.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Understand the Virtual Data Model

SAP HANA Live Virtual Data Model (VDM) Architecture

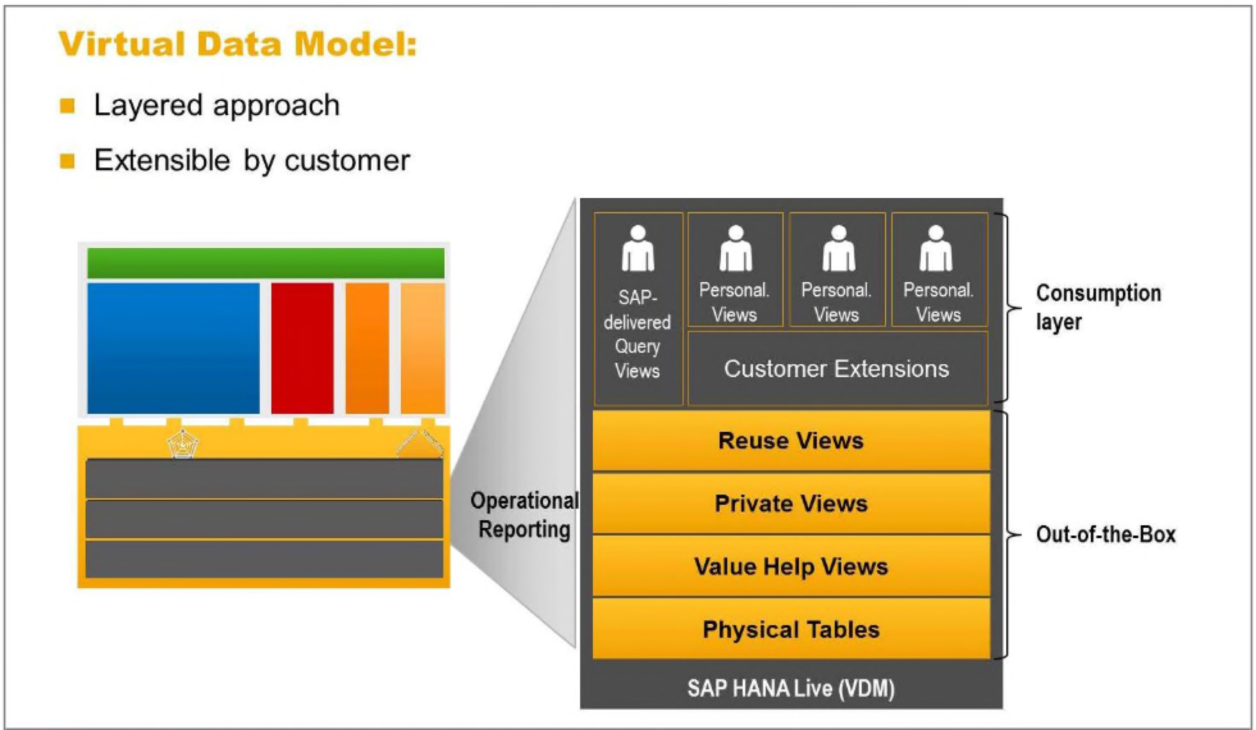


Figure 215: Detailed SAP HANA Live VDM Architecture

The Virtual Data Model (VDM) is a hierarchical structure that represents the SAP HANA database views used in SAP HANA Live. The VDM is built, maintained, and supported by SAP and follows consistent, and standard SAP HANA modeling rules. The VDM has been built especially with high performance on operational reporting in mind.

The virtual data model consists of the following types of views:

- Query views are designed for direct consumption by an analytical application or an analytical tool (for example, SAP BI tools). They are always the top layer in the hierarchy of views and are not designed for reuse in other views. SAP delivers query views to provide a quick start for customers who can use these immediately. However, the SAP supplied

query view only includes the attributes and measures chosen by SAP. They do not include all possible attributes and measures from the source tables. They also contain very few variables for user filter selection at run time. It is expected that customers will create their own custom query views, possibly by copying the standard query views and then apply their own filters, and variables.

- Reuse views are the heart of the virtual data model. They expose the business data in a well-structured, consistent, and comprehensible way covering all relevant business data in SAP Business Suite systems. They are designed for reuse by other views and must not be consumed directly by analytic tools. You must not make changes to the reuse views as updates to the VDM will overwrite your changes. Only SAP develop and maintain reuse views.
- Private views de-normalize the source database model, which is often far too complex for direct consumption. Private views are built on tables or other private views. You should never consume private views directly because they might not carry clear business semantics. They are intended to be used in reuse views. you should never modify private views as these are strictly developed and maintained by SAP and the changes you make will be overwritten when you apply updates to the VDM.
- Value Help views provide a full list of possible values for popular business entities (for example, material, and customers). The Value Help views are used to populate selection lists in business applications or reports to make filter choices easier by having all of the values visible. Value Help views are not consumed by VDM models, but can be consumed by applications directly.

Virtual Data Model

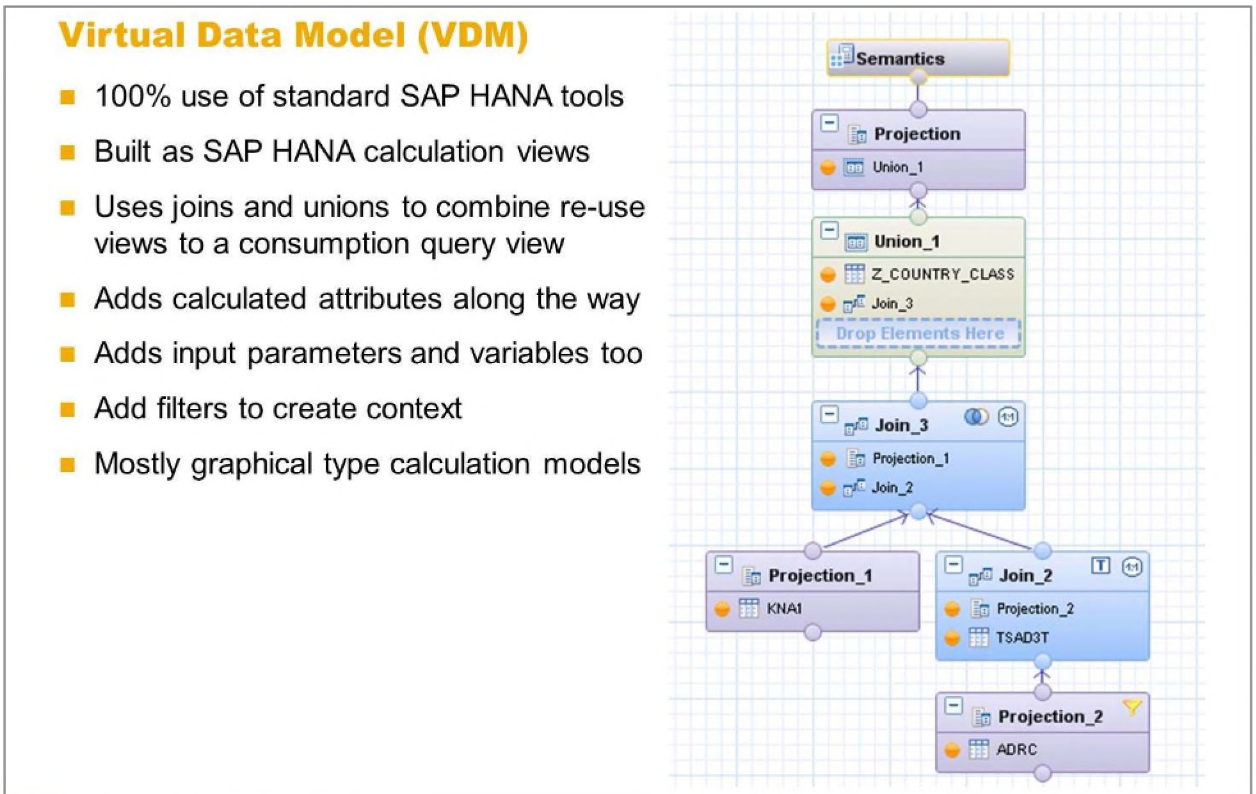


Figure 216: Virtual Data Model

In order to simplify modeling, graphical calculation views were chosen by SAP to be the default view type for SAP HANA Live. Graphical calculation views provide a high level of visibility for customers who can easily explore the supplied models. It also means that models

can be easily extended using standard graphical tools in SAP HANA studio. The only time a scripted calculation view is delivered is for the very few occasions when a modeling function is required that is not available in a graphical calculation view. But this is very rare and if they do exist then they are usually wrapped up with a graphical calculation view.

All SAP HANA Live graphical calculation views have the *Use SQL Engine* setting enabled. This is very important and it means that only the SQL Engine is used (not the Calculation, Join, or OLAP engines). The key reason for this is so that during execution the processor does not try to combine the different processing engines in the execution plan. If this happened, data would be passed back and forth between the engines, and as you have leaned earlier, this is not good for performance. SAP HANA Live models are developed with great care to use only features that are optimal for the SQL engine. We simply do not need the services of the other HANA engines, so we guarantee these engines are kept out of the execution by making the setting described above.

SAP HANA Live models expose operational data and by its very definition this means that the views are based on line level data. We do not aggregate data. Hence, we do not need the services of the OLAP engines and we do not see SAP HANA Live views using star joins in the graphical calculation views.

Text joins are used to support multi-language descriptions, but not within attribute views, only from the graphical calculation views.

Variables are extensively used but only in the query views to provide a run-time user selection. Input parameters are used in other views, but only where absolutely necessary to add required meaning to data. SAP tried to avoid input parameters as they are not always easily consumed by some SQL consumers, but if they are used then a default value is always supplied.

SAP HANA Live Calculation Views

You can create your own views of any type and combine these with delivered SAP HANA Live calculation views. The SAP HANA Live views can be complex with many layers. A key reason for the complexity is that we are de-normalizing the source data model. This means we are combining tables that were originally, and deliberately fragmented in the source system to aid OLTP performance.

Use the SAP HANA Live Browser to explore the model easily at a summary level. Also the SAP HANA studio can be used to investigate each model in more detail. When exploring the model with the SAP HANA studio pay attention to the icons in the nodes. These identify the type of source (another model or table). Also notice the yellow filter icon as this gives an indication that a node is filtering data that passes through it.

From any node highlight one of the source items and choose the *Open* menu option. This will open the source object and allows easy step by step navigation through the model.

Use the expand option in the SAP HANA studio modeler to open all nodes with one click so you can easily see all sources at a glance.

Unit 4
Exercise 19

Explore the SAP HANA Live Virtual Data Model

You are a system administrator, and you have been asked to explore the architecture of SAP HANA Live to develop your knowledge around the supplied Virtual Data Model.

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.

1. Launch the SAP HANA Studio and add your assigned system to the *Systems* view using the details supplied by the instructor or provided in your System Setup Guide.
2. Use the SAP HANA studio to locate the model *OpenSalesOrderQuery*. Explore this model to develop your understanding of how it was built by SAP.
3. Open the view *OpenSalesOrderQuery* and answer the following questions:

   What *type* of view is this?

   ____

   Are there any *variables* used?

   ____

   What is the purpose of the projection node *Projection_3*?

   ____

   Apart from exposing only limited columns, what other purpose does the projection node *Projection_1* have?

   ____

What is the purpose of the node *Join_2*?

How can you be sure this view always executes in the SQL engine?

Unit 4
Solution 19

Explore the SAP HANA Live Virtual Data Model

You are a system administrator, and you have been asked to explore the architecture of SAP HANA Live to develop your knowledge around the supplied Virtual Data Model.

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.

1. Launch the SAP HANA Studio and add your assigned system to the *Systems* view using the details supplied by the instructor or provided in your System Setup Guide.
2. Use the SAP HANA studio to locate the model *OpenSalesOrderQuery*. Explore this model to develop your understanding of how it was built by SAP.
   a) Navigate to the required model by expanding the nodes: *Content → sap → hba → ecc → Calculation Views*.

   > Hint:
   > Use the filter menu option on the package to reduce the result set of views by typing all or just a partial name of the view. In this case you can enter **open**, so you see only views containing this word.

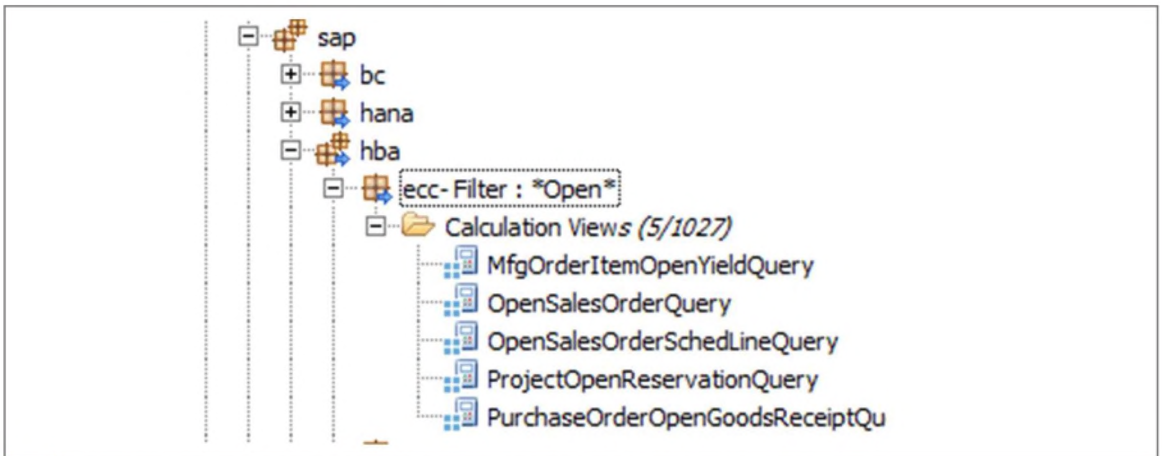


Figure 217: Locating SAP HANA Live Views Using a Filter

3. Open the view *OpenSalesOrderQuery* and answer the following questions:

   What *type* of view is this?

   Graphical calculation view

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

Are there any *variables* used?

Yes, a variable is used to select the Client number.

What is the purpose of the projection node *Projection_3*?

The purpose of the projection node Projection_3 is to provide two additional calculated measures.

Apart from exposing only limited columns, what other purpose does the projection node *Projection_1* have?

This node filters the Order Process Status 'A 'and 'B'

What is the purpose of the node *Join_2*?

The purpose of the node Join_2 is to provide the name of the Sales Office.

How can you be sure this view always executes in the SQL engine?

Select the *View Properties* tab and review the *Execute In* setting.

LESSON SUMMARY

You should now be able to:

- Understand the Virtual Data Model

Unit 4
Lesson 3

Discovering and Consuming SAP HANA Live Views

LESSON OVERVIEW

SAP deliver a huge number of useful SAP HANA Live views. In this lesson, you will begin by learning how to discover the SAP HANA Live views, and then later, you will learn how to consume the SAP HANA Live views.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Discover and consume SAP HANA Live views

Discovering SAP HANA Live Views

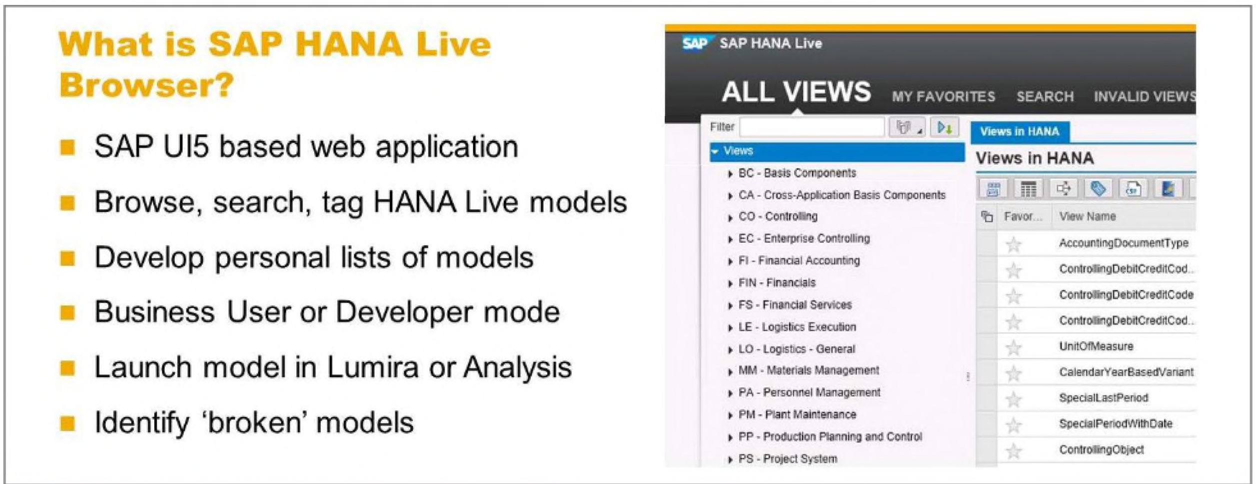


Figure 218: SAP HANA Live Browser - Description

SAP HANA Live Browser is an SAPUI5 web application that allows you to easily and quickly browse, search, and tag SAP HANA Live views and try them out in SAP Lumira or SAP BusinessObjects Analysis, edition for Microsoft Office.

With the SAP HANA Live Browser, you can browse SAP HANA Live views organized by Application Component or browse SAP HANA Live views organized by SAP HANA modeling packages. This makes is easy to find the views you are looking for.

This tool allows you to search for views. The search considers not just the title of the view but even the column names within the views. So, for example, you could easily locate all SAP HANA Live views that contain the column 'order reason'.

SAP HANA Live views are always dependent on underlying views and tables being available (not just present but also active). Broken models (listed under the tab Invalid Views) are those SAP HANA Live views where active, dependent models or source tables are missing. This

means that the SAP HANA Live view cannot be activated and therefore it cannot be used and sits under Invalid Views until its dependent objects are made available.

SAP HANA Live Browser is also able to display all customer-created views, not just those belonging to SAP HANA Live. So you can use the tool as a help to catalog and explore your own views. Custom views appear under the *UNDEFINED* node in the Application Component hierarchy. Your custom views are not assigned to SAP Application Components, but you can still browse your custom views by SAP HANA packages.

Architecture

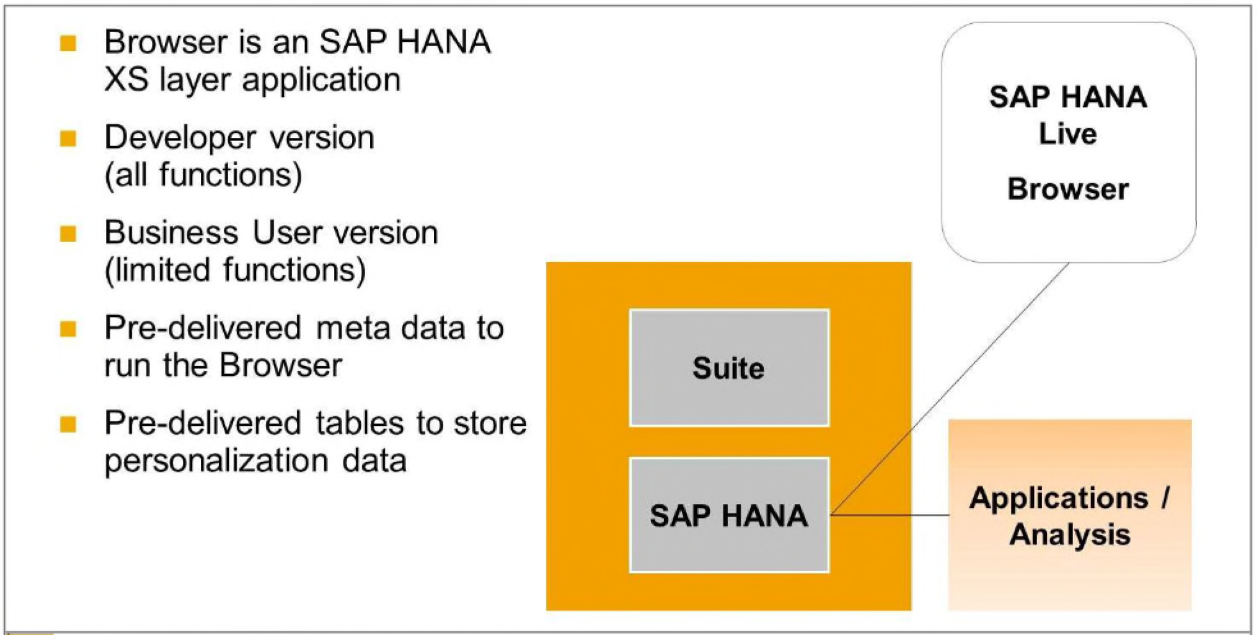


Figure 219: SAP HANA Live Browser - Architecture

The application can be used by anyone who wants to explore SAP HANA Live views without using SAP HANA studio. One of the unique features of SAP HANA Live Browser is the ability to obtain a complete and easy to interpret overview of an entire SAP HANA Live model including all views and tables that are involved. This is lineage at the view/table level. This cannot easily be done with the SAP HANA studio. You cannot view lineage at the column level using SAP HANA Live Browser, so to trace the journey of a column through a model you would need to use SAP HANA studio, which provides a more detailed view of a model.

SAP HANA Live Browser is a tool in the SAP HANA Live family and must be installed separately. The installation procedure is documented at help.sap.com. The basics steps include downloading and installing the package (this provides the XS application code to run the tool and once installed you can see a new package (sap > hba > explorer). Also the schema (SAP_EXPLR) where the tables and views used by the tool (to search, manage tags, and favorites.) are located. When you have fully installed the tool you are ready to logon to the SAP HANA Live Browser.

Versions of SAP HANA Live Browser

There are two versions of SAP HANA Live Browser. A full feature version to support the developer and also a limited feature version to support the business user who does not need all technical options.

To use a version, you must assign either the developer or business role (or both) to an SAP HANA user.

The versions of SAP HANA Live Browser are as follows:

1. Business User version: The user is able to explore models, preview data, and launch the view in SAP Lumira or Analysis.

   - **`http://<SAP HANA Server Host>:80<SAP HANA Instance>/sap/hba/explorer/buser.html`**
   - Technical name of the required business user role: **`sap.hba.explorer.roles::Business`**

2. Developer version: In addition to the business user version, the developer is also able to add additional metadata to models, such as tags to aid searching, view broken models and create a list of tables needed for replication in a side car deployment.

   - **`http://<SAP HANA Server Host>:80<SAP HANA Instance>/sap/hba/explorer`**
   - Technical name of developer role: **`sap.hba.explorer.roles::Developer`**

> Note:
> The official name for this tool is SAP HANA Live Browser. But often you hear the tool incorrectly referred to as SAP HANA Live Explorer. This probably originates from the technical component name which is HCO_HBA_EXPLORER.

Features of SAP HANA Live Browser

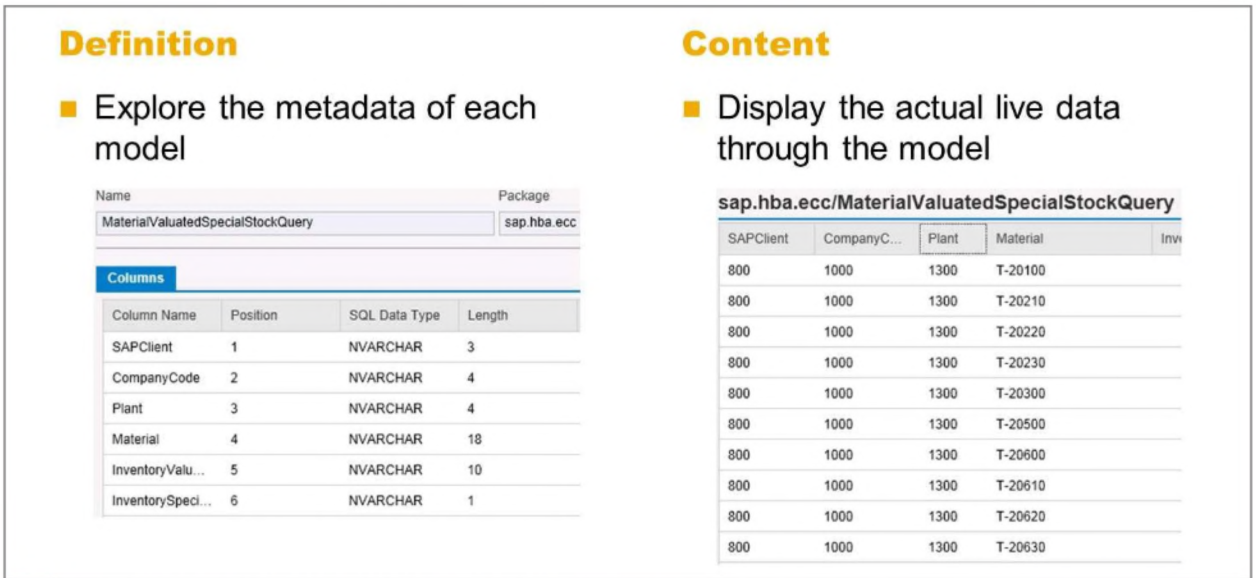


Figure 220: Explore the Metadata and Content of a Model

To preview metadata in an open view use *Open Definition*, and to preview data use the *Open Content* tab.

Use the *Search* tab in the top menu to search for views, tables, view description, view column names, or private tags. When you expand a view, matched tags, tables, and columns display. Upon enabling the auto complete feature, the name of the view displays. You can also choose to view the definition or content of each view depending on your analytical privileges.

When you enter a word, views that consume the table directly or indirectly are listed. You can enter the following set of characters to search: **`a-z, A-Z, 0-9, &, +, _`** and space.

Explore the Object Relationships and Generate List of Tables for a Model

For efficiency, SAP HANA Live views display in a tree view format on the left side. When you select any view, details of the respective view display on the right side. For views that are open, you can choose to view metadata (*Open Definition* tab) or Data (*Open Content* tab), as well as choose to view tables and views used as cross references (*Cross Reference* tab).

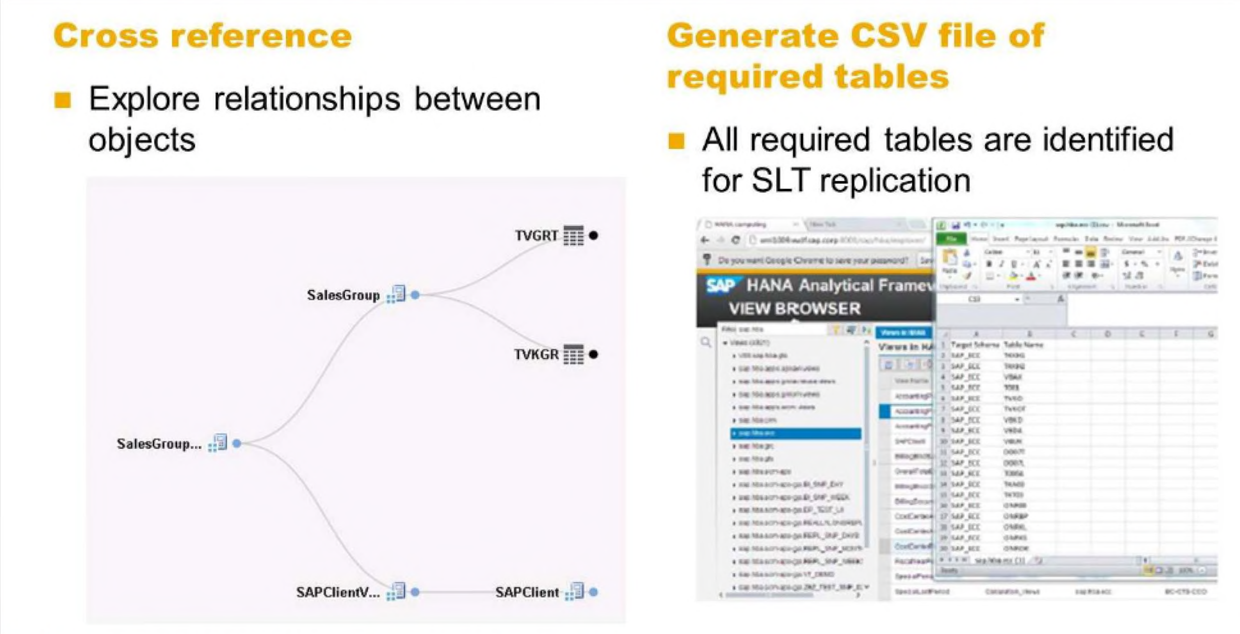


Figure 221: SAP HANA Live Browser - Features

To display all views and tables within a model, select the *Open Cross Reference* tab. You can then choose *Graph View* to view cross references as a graph. All the information can be exported in XML format.

You can quickly generate a CSV file for a view or entire package or application component by right-clicking the respective package, application component and then choosing *Generate SLT file*. You can also choose *CSV* on the top menu. The SLT file option generates a CSV file containing a list of the tables that are needed by the SAP HANA Live views.

One of the benefits of choosing to use SLT Replication Server to replicate source suite tables in a side car deployment, is that you can import the generated list of tables directly into SLT Replication Server. This means that you do not need to manually enter each table name in the replication console, and this list can be very long.

To generate connection files for SAP Lumira, select a view from the SAP HANA Live Browser and choose *Open View in SAP Lumira*. This feature generates a .svic file. If you open the file, you are prompted to enter the username and password to grant access to the SAP HANA system.

To open the view in Analysis, Edition for Office, select a view from the SAP HANA Live Browser and choose *Open View in Analysis Office*.

Unit 4
Exercise 20

Discover SAP HANA Live Views

Business Example

You want to explore the models that are provided with SAP HANA Live using a simple-to-use web browser based interface.

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.

1. Log on to the HANA Live Browser, developer version, by using the following URL and your SAP HANA credentials:

   `http://wdflbmt7215.wdf.sap.corp:8000/sap/hba/explorer/`

   > Note:
   > A shortcut to this URL is available in your SAP HANA Student folder *Student → HA300*.
   >
   > The shortcut is called *SAP HANA Live Browser (Developer).url*.

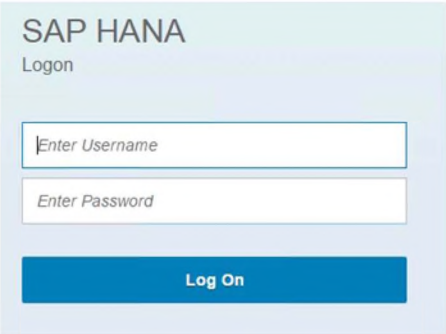
   

   Figure 222: Logon Screen

2. Using only the Application Component view, locate the query view: *OpenSalesOrderQuery*

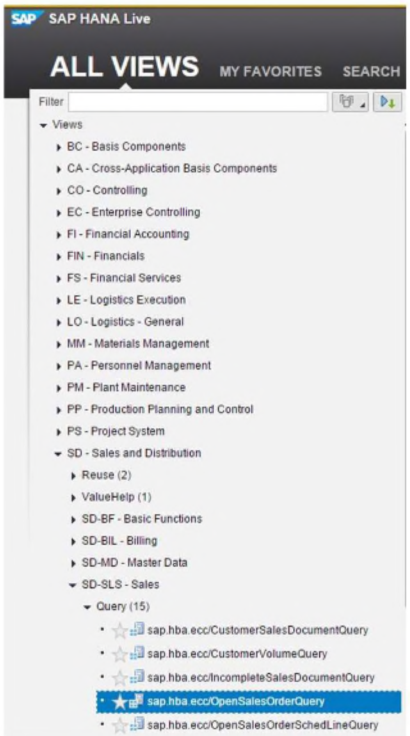


Figure 223: Application Component Hierarchy

3. Highlight the selected view and choose *Open Definition* in the toolbar to explore the structure of the query.

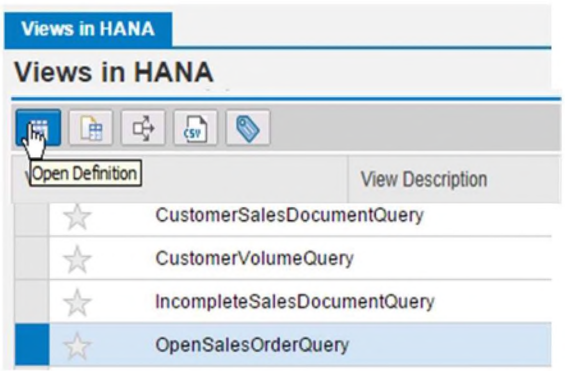


Figure 224: Opening View Definition

4. Explore the metadata of the view.

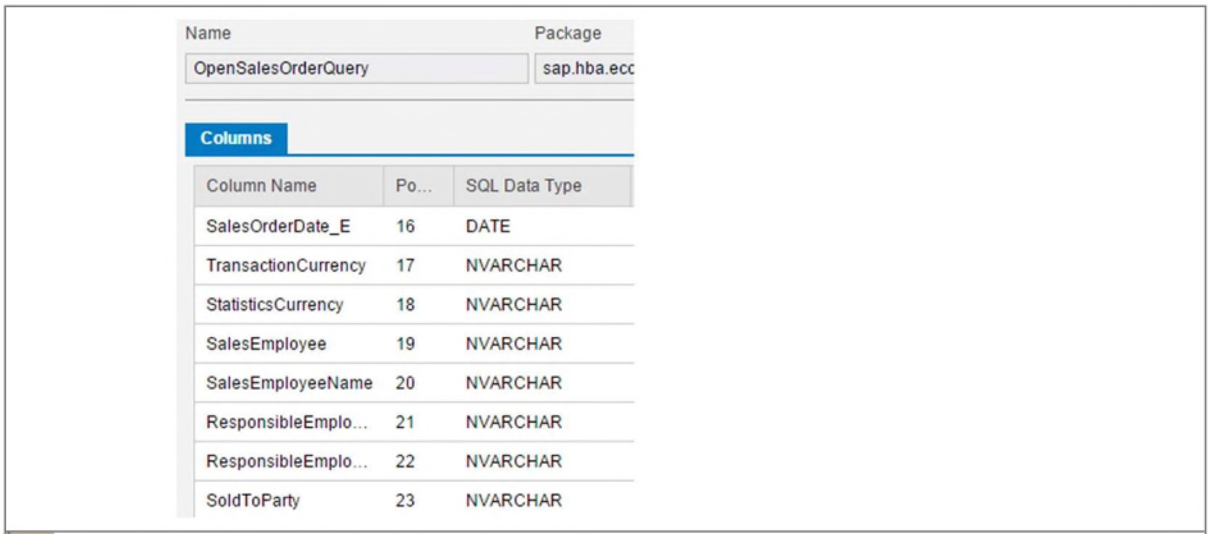


Figure 225: Exploring Metadata

5. Choose *Open Content* to view the business data that is exposed through the selected view.

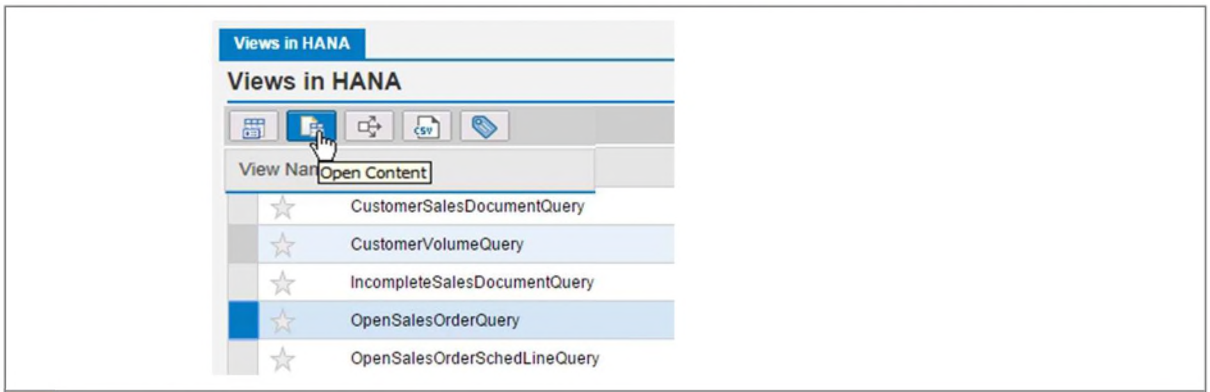


Figure 226: Opening View Content

6. At the prompt for *Client*, enter **800** and choose *OK*.

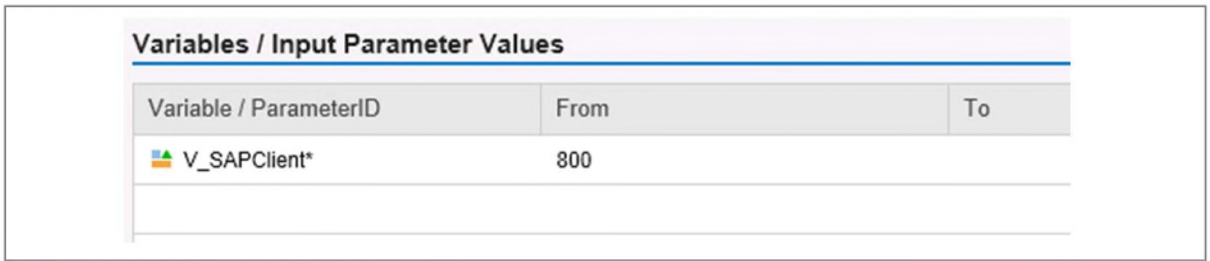
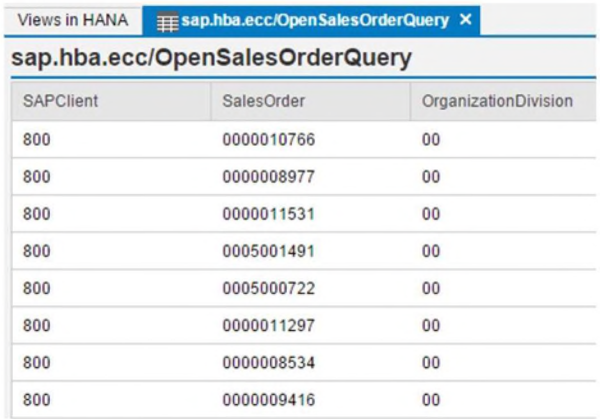


Figure 227: Provide Prompt Values

7. Explore the data exposed.



SAPClient	SalesOrder	OrganizationDivision
800	0000010766	00
800	0000008977	00
800	0000011531	00
800	0005001491	00
800	0005000722	00
800	0000011297	00
800	0000008534	00
800	0000009416	00

Figure 228: Exposed Data

8. Choose *Open Cross Reference*, and then choose *Graph View*.

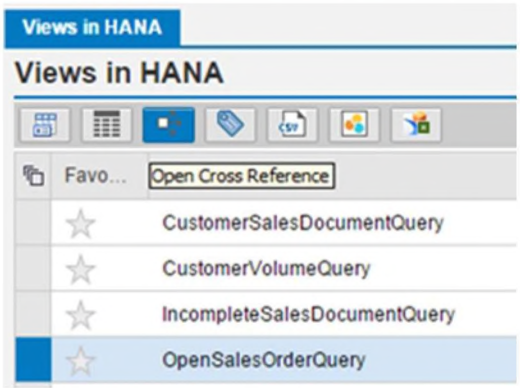


Figure 229: Opening the Cross Reference

9. Display the related views and tables and their dependencies for the selected SAP HANA Live view.

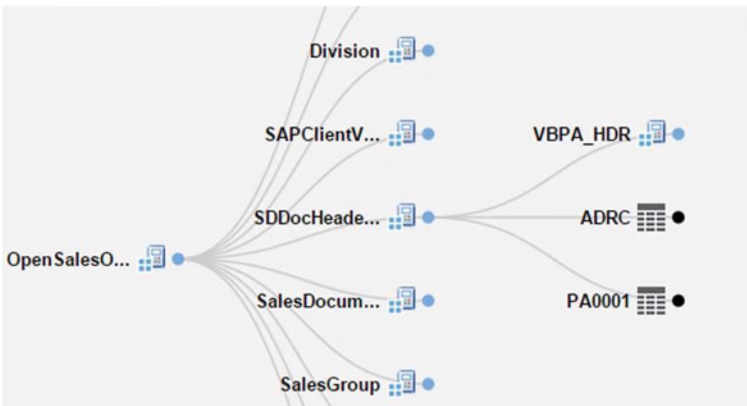


Figure 230: Graph View of the Model

10. Add a tag to the view so that you are able to search for this more easily using familiar business words. Use the tag word **`supervisor`**.

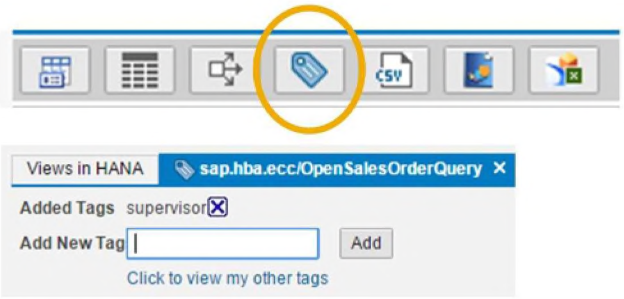


Figure 231: Adding a Tag

11. Locate the same view but use the *Search* feature so that you can enter the name of the tag you created in the previous step.

    Notice how your search returns many views and not just the *OpenSalesOrderQuery* view. One reason is that several views have columns names that include the word *supervisor*.

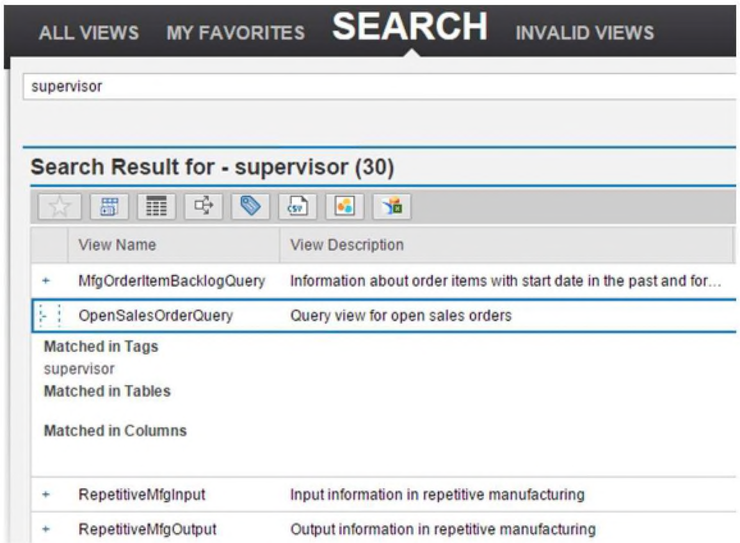


Figure 232: Results of Search Using Tags

12. Add the *OpenSalesOrderQuery* view to your Favorites so it is easy to locate.

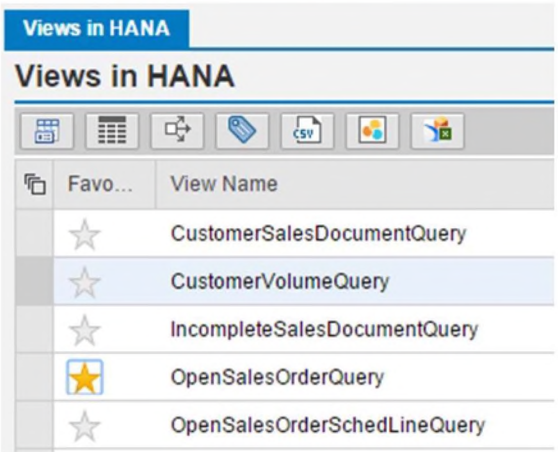


Figure 233: Marking a View as a Favorite

13. Navigate to the *MY FAVORITES* screen to see your view again.

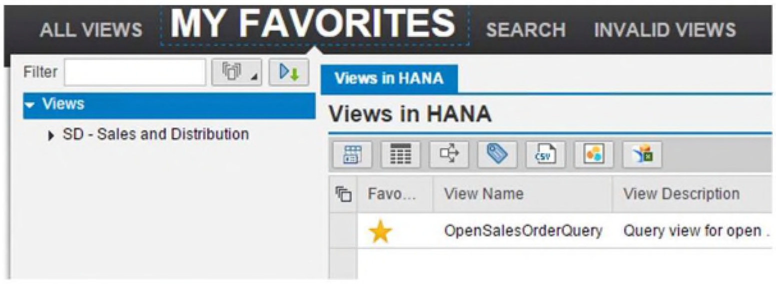


Figure 234: View Favorites

Unit 4
Solution 20

Discover SAP HANA Live Views

Business Example

You want to explore the models that are provided with SAP HANA Live using a simple-to-use web browser based interface.

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.

1. Log on to the HANA Live Browser, developer version, by using the following URL and your SAP HANA credentials:

   `http://wdflbmt7215.wdf.sap.corp:8000/sap/hba/explorer/`

   > Note:
   > A shortcut to this URL is available in your SAP HANA Student folder *Student → HA300*.
   >
   > The shortcut is called *SAP HANA Live Browser (Developer).url*.

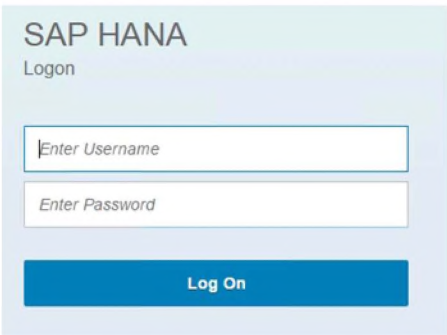
   

   Figure 222: Logon Screen

   a) In a web browser, go to **`http://wdflbmt7215.wdf.sap.corp:8000/sap/hba/explorer/`**.

   b) Alternatively, double-click the corresponding shortcut from your SAP HANA Student folder.

2. Using only the Application Component view, locate the query view: *OpenSalesOrderQuery*

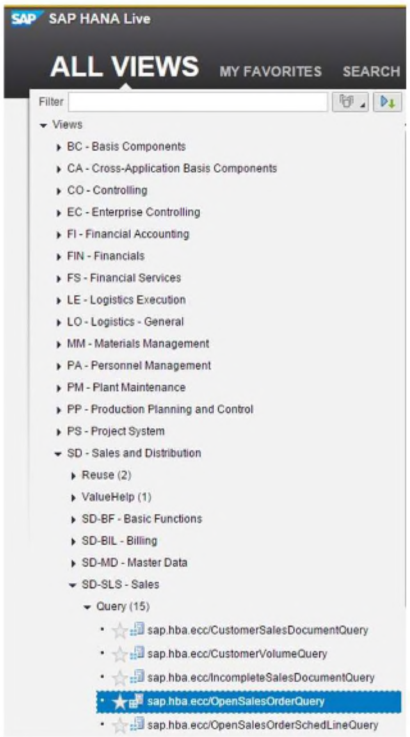


Figure 223: Application Component Hierarchy

a) Go to *SD* → *SD-SLS* → *Query*.

> Note:
> You can also use the filter field to locate the view.

3. Highlight the selected view and choose *Open Definition* in the toolbar to explore the structure of the query.

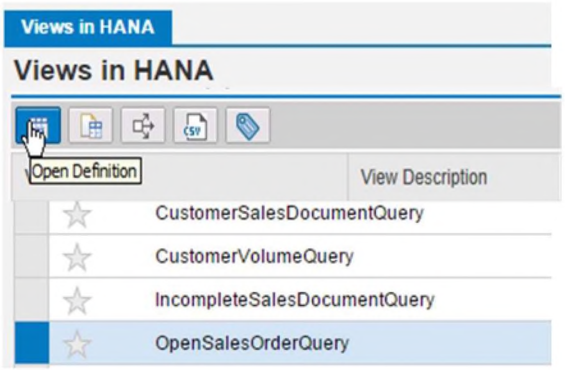


Figure 224: Opening View Definition

4. Explore the metadata of the view.

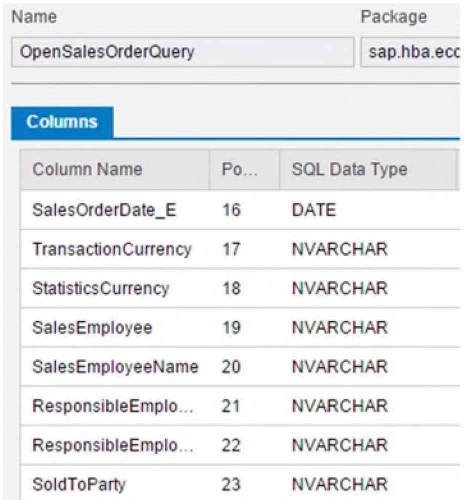


Figure 225: Exploring Metadata

5. Choose *Open Content* to view the business data that is exposed through the selected view.

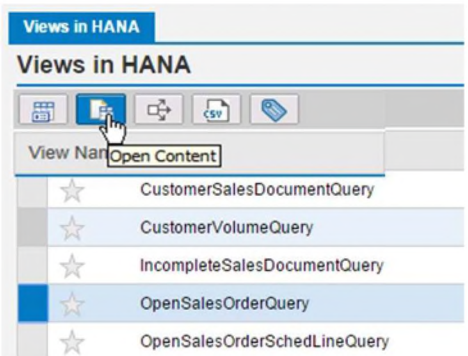


Figure 226: Opening View Content

6. At the prompt for *Client*, enter **800** and choose *OK*.

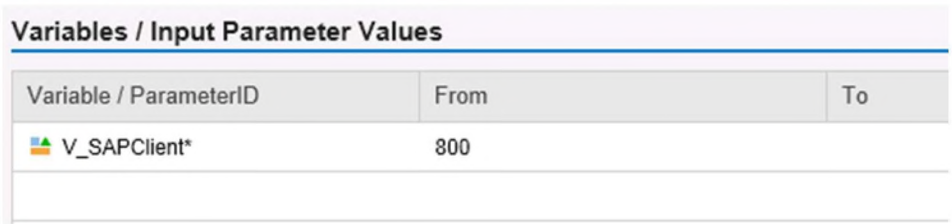
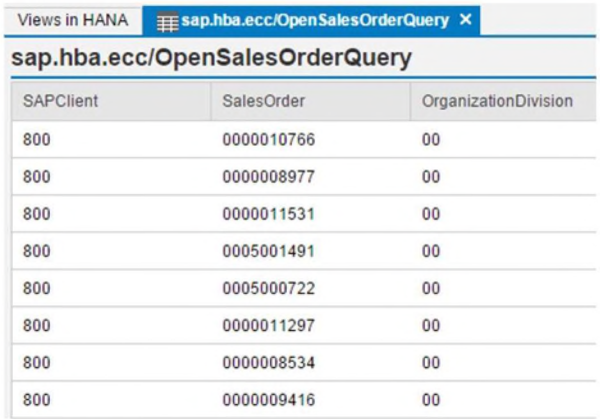


Figure 227: Provide Prompt Values

7. Explore the data exposed.


Views in HANA | sap.hba.ecc/OpenSalesOrderQuery

sap.hba.ecc/OpenSalesOrderQuery

SAPClient	SalesOrder	OrganizationDivision
800	0000010766	00
800	0000008977	00
800	0000011531	00
800	0005001491	00
800	0005000722	00
800	0000011297	00
800	0000008534	00
800	0000009416	00

Figure 228: Exposed Data

8. Choose *Open Cross Reference*, and then choose *Graph View*.

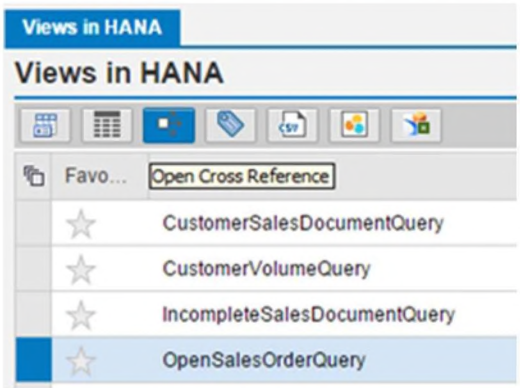


Figure 229: Opening the Cross Reference

9. Display the related views and tables and their dependencies for the selected SAP HANA Live view.

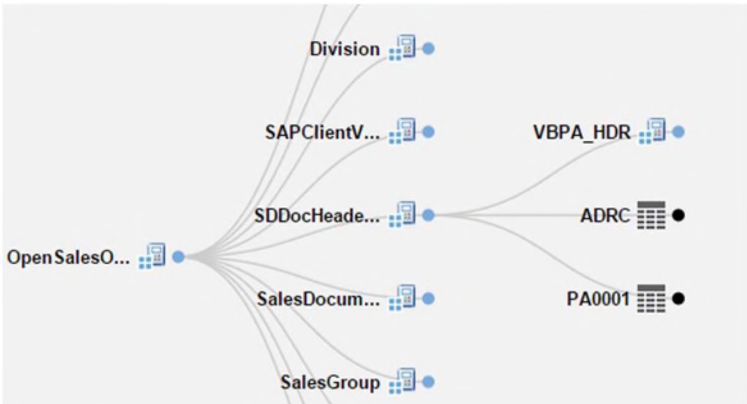


Figure 230: Graph View of the Model

a) Choose *Graph View* to see the dependencies shown graphically.

b) Expand all nodes to view all dependent objects and then hover over the nodes to see the full name of the view or table.

10. Add a tag to the view so that you are able to search for this more easily using familiar business words. Use the tag word **`supervisor`**.

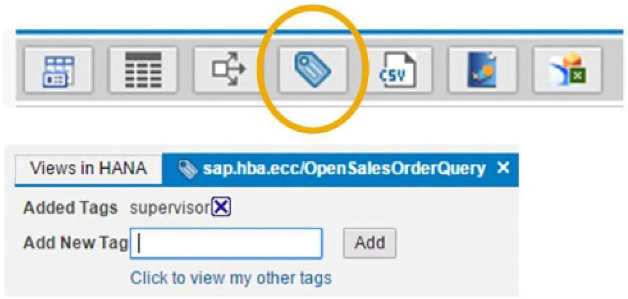


Figure 231: Adding a Tag

11. Locate the same view but use the *Search* feature so that you can enter the name of the tag you created in the previous step.

    Notice how your search returns many views and not just the *OpenSalesOrderQuery* view. One reason is that several views have columns names that include the word *supervisor*.

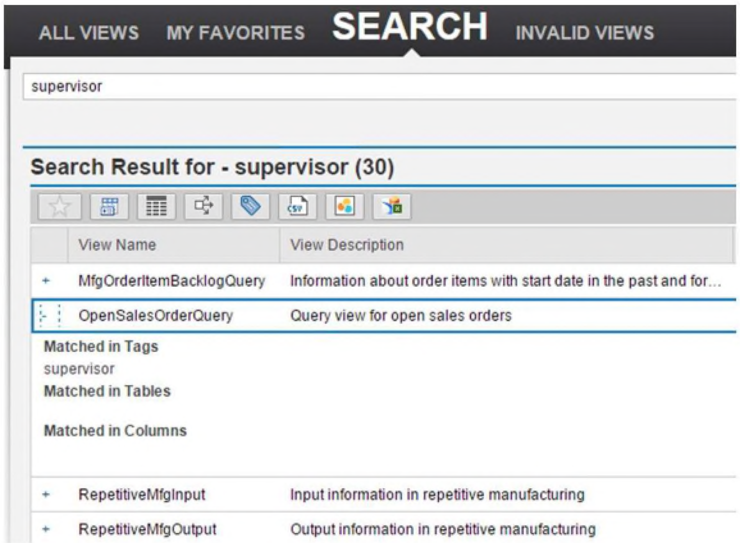


Figure 232: Results of Search Using Tags

12. Add the *OpenSalesOrderQuery* view to your Favorites so it is easy to locate.

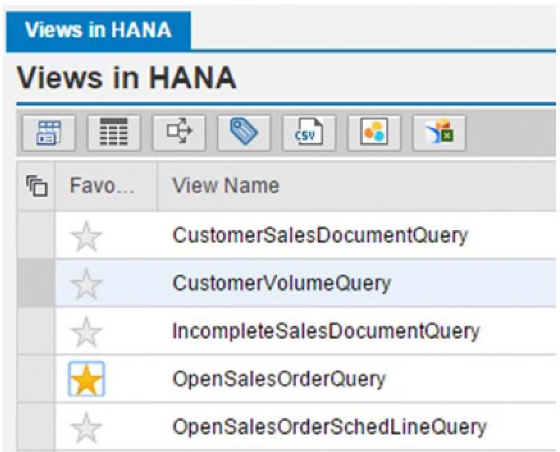


Figure 233: Marking a View as a Favorite

a) In the *ALL VIEWS* screen, select the *Views in HANA* tab.

b) Select the gray star shaped icon, which changes to orange when selected.

c) Alternatively, from the *SEARCH* screen, select the view and choose the top-left button *Add to favorites*.

13. Navigate to the *MY FAVORITES* screen to see your view again.



Figure 234: View Favorites

Consuming SAP HANA Live Views

SAP HANA Live views are technically no different to any other views in SAP HANA, whether created by SAP, partners, or customers. As with any SAP HANA view, all SAP BI reporting tools are able to consume SAP HANA Live views. SAP HANA Live views frequently contain variables and these are consumed by each SAP BI reporting tool. This means you are able to provide pop-up filters to the users to make personal data selections. You could also add custom variables too, this will be covered in a later lesson.

The figure, What is SAP BI?, lists the SAP BI reporting tools and their purposes. This is just for background information.

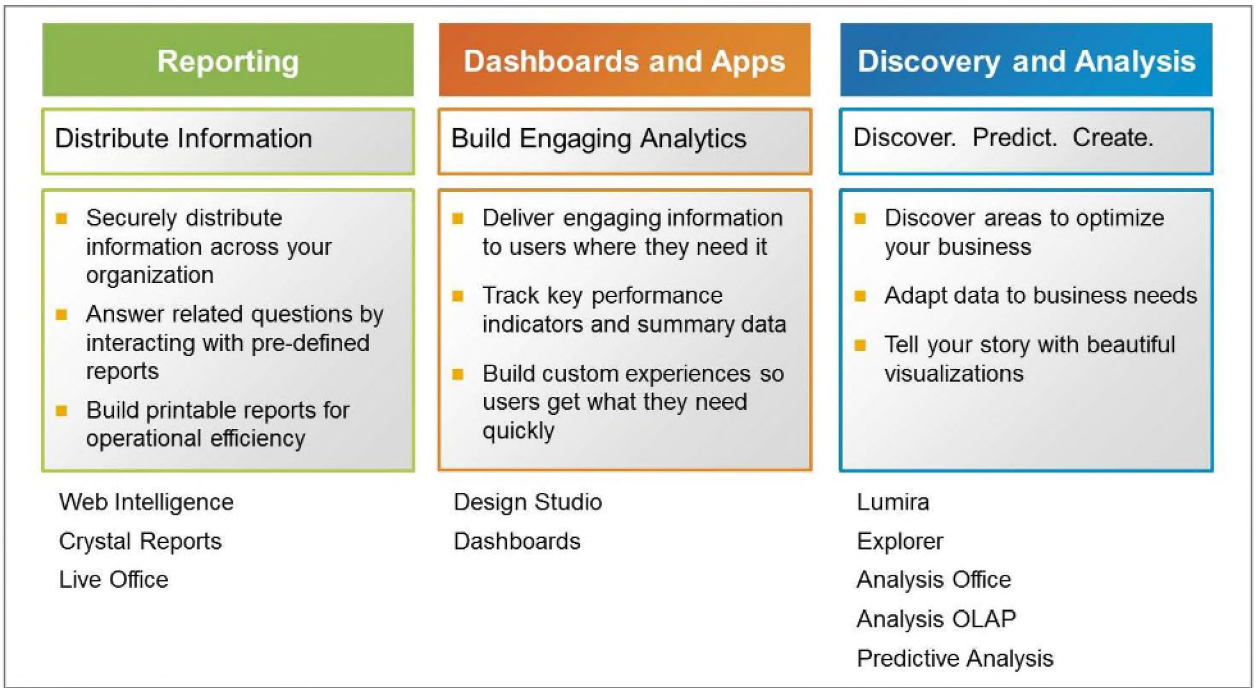


Figure 235: What is SAP BI?

There are a number of ways to connect SAP BI reporting tools to SAP HANA. Regardless of whether you use SAP HANA Live views or custom views it is important to understand how each reporting tool consumes SAP HANA views and then choose the reporting tool that fits your requirements.

Not all SAP BI reporting tools are able to consume the same content from SAP HANA. For example, not all reporting tools can consume all types of hierarchy created in an SAP HANA model.

Refer to the detailed online documentation (help.sap.com) for details on which SAP HANA features are compatible with each SAP BI reporting tool.

> Note:
> The SAP training course HABO1 — SAP BI Integration with SAP HANA, covers in detail the SAP HANA features that are compatible with each SAP BI tool. The course also covers the best practices for report design using SAP BI on SAP HANA.

To accelerate the implementation of SAP BI based reports, SAP have developed a Rapid Deployment Solution (RDS) which delivers many pre-defined reports based on various tools.

Rapid Deployment Solutions for SAP HANA Live

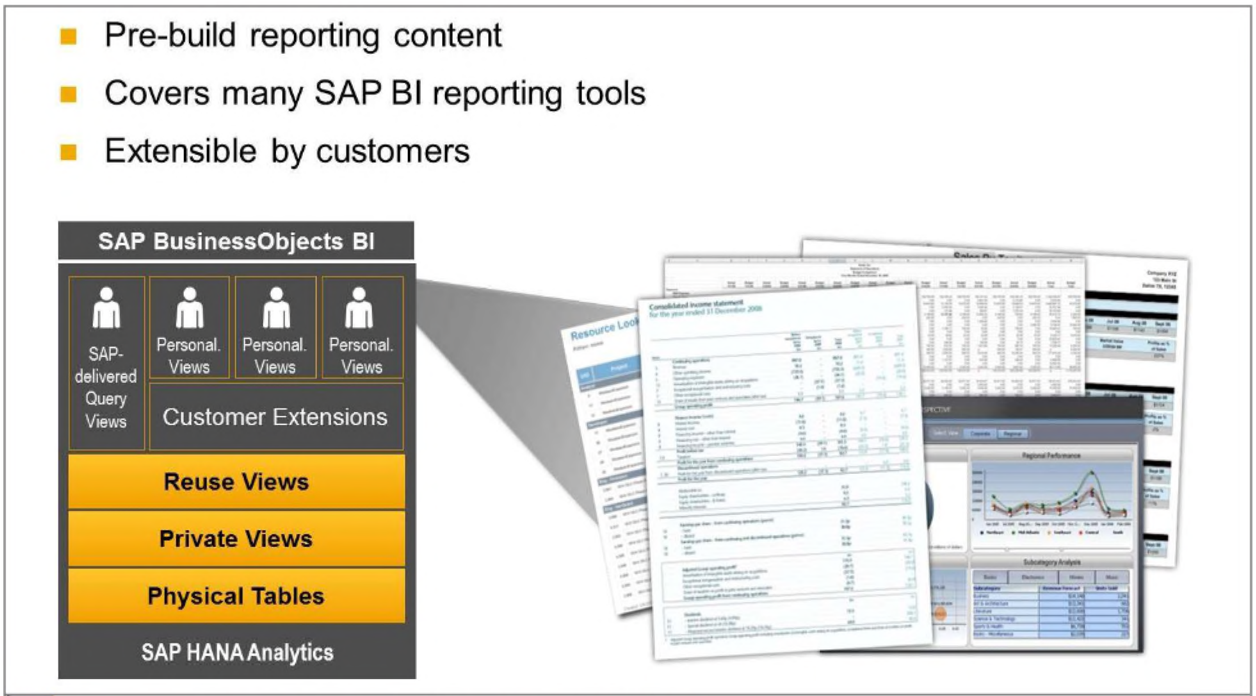


Figure 236: Rapid Deployment Solutions for SAP HANA Live

RDS for SAP HANA Live includes ready-made reports for various SAP BI reporting tools such as SAP Crystal Reports, and Dashboards. RDS for SAP HANA Live is a separate purchase and installation and is not included with SAP HANA Live.

Real-Time Analytical Applications

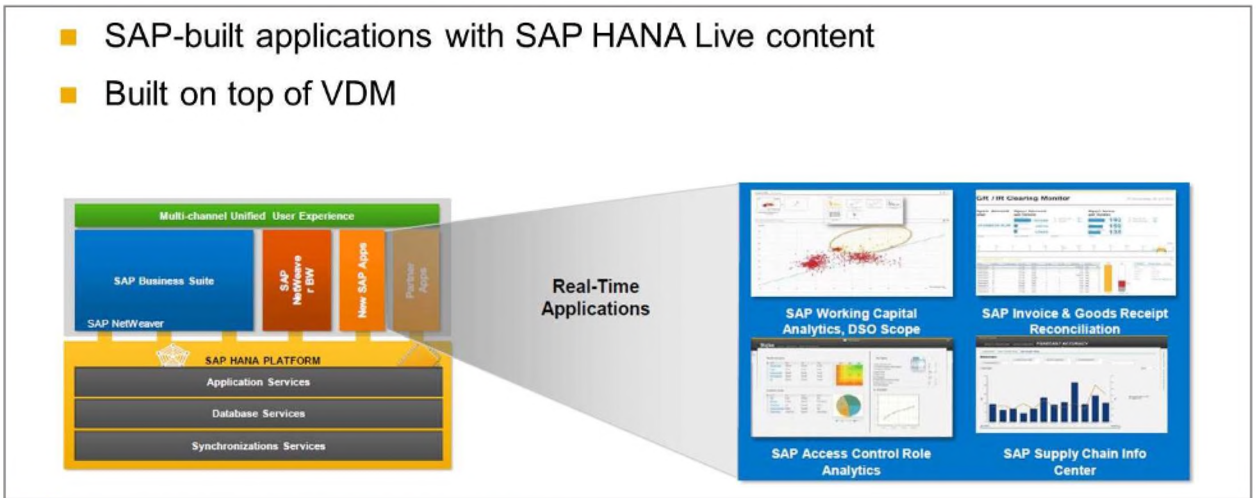


Figure 237: Real-Time Analytical Applications

There are many delivered real-time applications that are included with SAP HANA Live. They are created using the SAPUI5 development kit (HTML5, OData, SAP NetWeaver Gateway Services) to create a consumer grade experience. Also referred to as Smart Business Cockpits, the applications rely heavily on the VDM delivered with SAP HANA Live.

Goods Receipts / Invoice Receipts Clearing

- Improve clarification process for invoices
- Avoid paying unjustified vendor invoices
- Foster communication between different departments
- Quickly retrieve the purchasing history, receipts by vendor and contact data for clarification. Don't get lost in different screens.
- Designed for the web browser and the mobile device
- Powered by SAP HANA on real-time ERP data



Figure 238: Goods Receipts / Invoice Receipts Clearing

Goods Receipts / Invoice Receipts Clearing is an example of an SAP HANA Live application that sits on top of the VDM.

This is part of standard SAP HANA Live (not SAP HANA Live RDS).

Days Sales Outstanding

- Analyze drivers of your company's days sales outstanding (DSO)
- Allows to interactively build up flexible analysis paths to analyze data
- Innovative visualization of analysis process sequence
- Supports direct calculation of relevant KPIs based on original documents, leading to more accurate figures than indirect calculations based on financial projections
- Designed for iPad
- Powered by SAP HANA on real-time ERP data

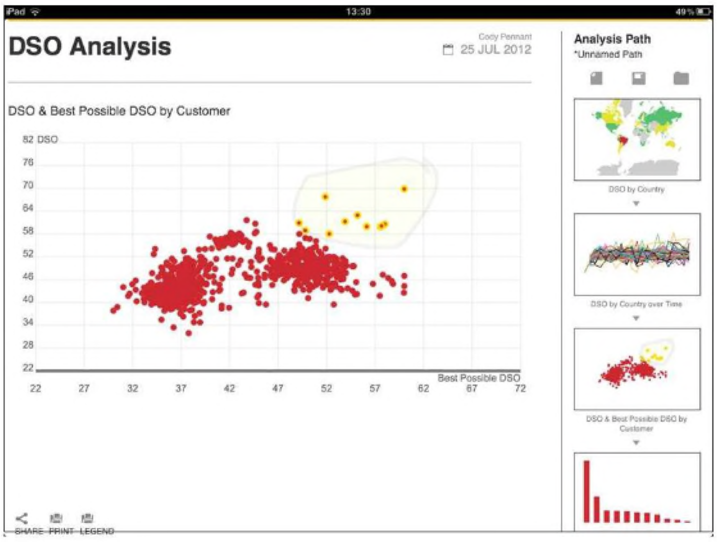


Figure 239: Days Sales Outstanding

Days Sales Outstanding is an example of an SAP HANA Live application that sits on top of the VDM.

This is also part of SAP HANA Live (not the RDS).

GRC Analytics - Overview

- Analyze roles that have not been used for a period of time and de-provision users for these roles
- Analyze roles that have not been included in other roles or assigned to any users for a period of time, and de-commission these roles
- Analyze roles that have been frequently used for a period of time and initiate risk analysis run in the GRC Access Control core application
- Designed for the web browser and the mobile device

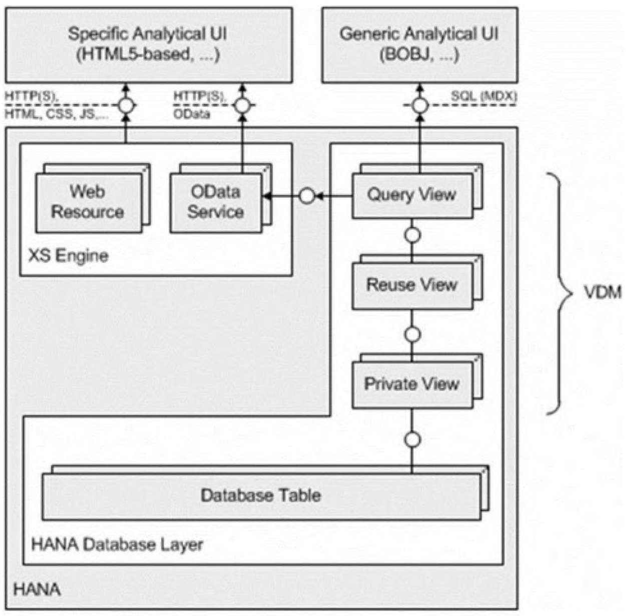


Figure 240: GRC Analytics - Overview

GRC Analytics - Overview is an example is of an SAP HANA Live application that sits on top of the VDM.

This is also part of SAP HANA Live (not the RDS).

Overview of Consumption Architecture



Consumption Examples

Business Objects Reporting Suite

Any SQL/MDX reporting tools

SAP BW (live ODS)

Desktop applications

Mobile applications

Figure 241: Overview of Consumption Architecture

Consumption by SAP BI is a popular use case for SAP HANA Live and provides a simplified, 2–tier architecture for live, operational analytics. However, this is not the only use case for SAP HANA Live. Any SAP HANA application developer can easily consume an SAP HANA Live view either directly (SQL) or through OData services. SAP S/4HANA relies heavily on SAP HANA Live views and many SAP Fiori applications and Smart Business Cockpits consume them.

Unit 4
Exercise 21

Consume SAP HANA Live Views

Business Example

As a Business user, you want to use Analysis Edition for Office in order to consume an SAP HANA Live view. This view displays sales orders that have not yet been fully processed, and allows you to aggregate the view to sum the total value of money outstanding by various analysis dimensions.

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.

1. Start Analysis for Microsoft Excel from the Windows Start Screen.

2. From the toolbar, select the *Analysis* tab, and from the ribbon, choose *Insert* → *Select Data Source*.

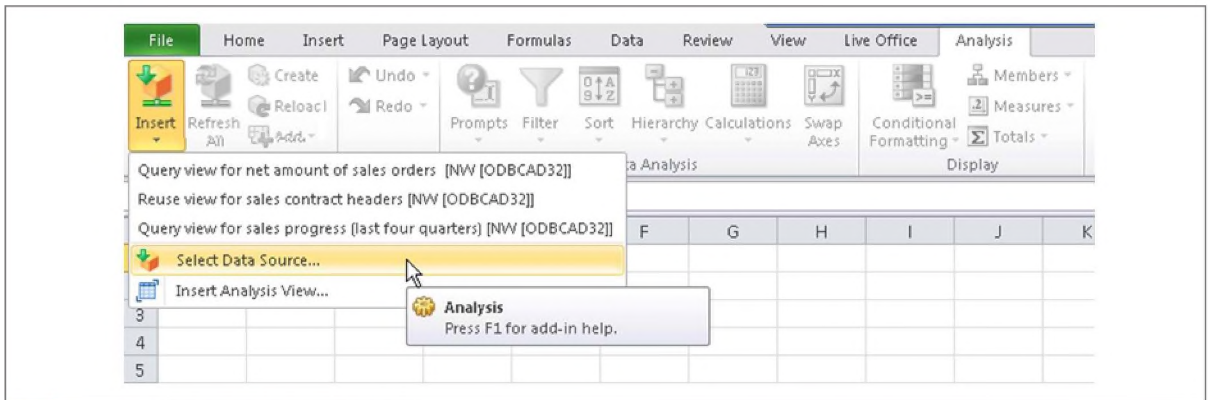


Figure 242: Data Source Analysis

> Note:
> If a *Logon to SAP BusinessObjects BI Platform* screen opens, choose *Skip*.

3. In the *Select Data Source* screen, show the connections from the Local System, choose the *HANA_##* data source and choose *Next*.

> Note:
> If another data source is already selected, choose *Back* to check the selected data source.

4. If the *HANA_##* connection does not exist, create it as follows:

- Right-click anywhere in the connections list and from the context menu, choose *Create new SAP HANA Connection*.
- Enter the following details:

Field	Value
Description	**HANA_##**
Scheme	*HTTP*
Host Name	**http://wdflbmt7215.wdf.sap.corp**
Port	**8000**
Authentication type	*Basic*

- Choose *Create and Logon*.

5. Enter the logon information from the table below and choose *OK*.

Field	Value
User	**STUDENT##** (## is your group number)
Password	**Training1**

6. Select the view type *Cube*, and in the *Search For* field, enter **opensales** and press Enter.
7. Select the *Query view for open sales orders* data source and choose *OK*.

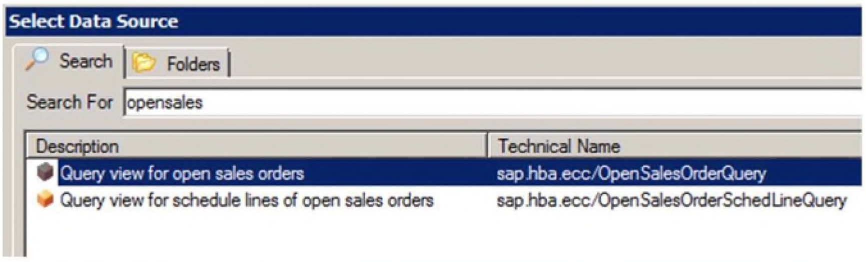


Figure 243: Locate the SAP HANA Live View

8. At the variable prompt for SAP Client enter **800** and choose *OK*.

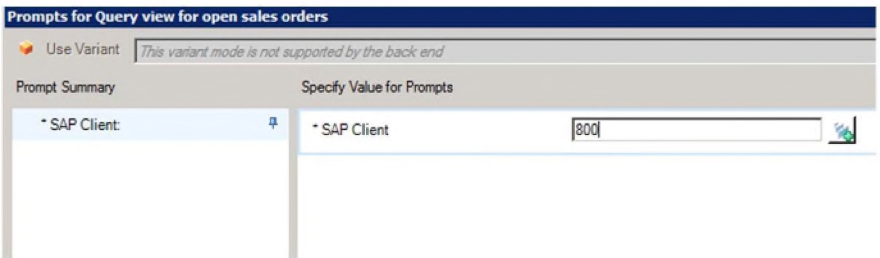
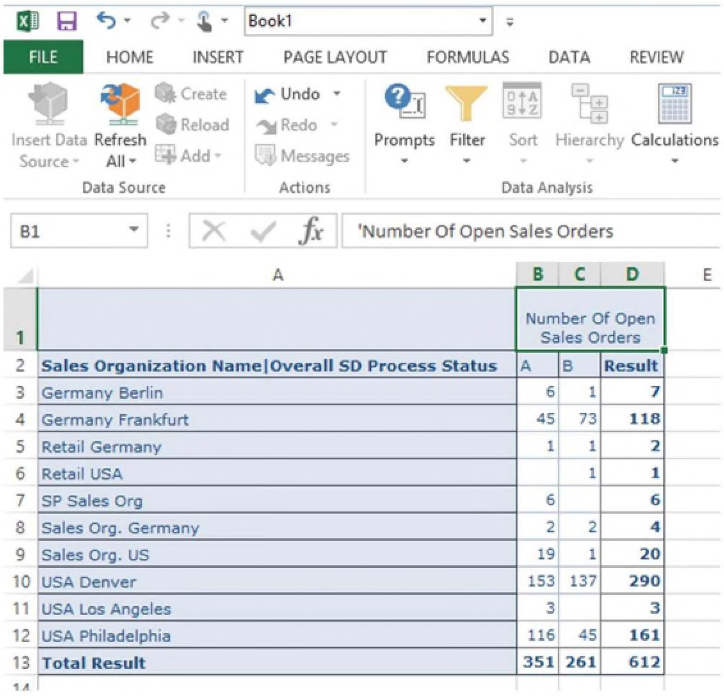


Figure 244: Complete the Prompt Entry

9. Drag the *Sales Organization Name* field to the *Rows* pane.
10. Restrict the list of measures to show only *Number of Open Sales Orders*.

    Right-click *Measures* in the *COLUMNS* pane, choose *Filter Members* and deselect the unwanted measure.

11. Add the attribute *Overall SD Process Status* to the *COLUMNS* pane.



Sales Organization Name|Overall SD Process Status	A	B	Result
Germany Berlin	6	1	7
Germany Frankfurt	45	73	118
Retail Germany	1	1	2
Retail USA		1	1
SP Sales Org	6		6
Sales Org. Germany	2	2	4
Sales Org. US	19	1	20
USA Denver	153	137	290
USA Los Angeles	3		3
USA Philadelphia	116	45	161
Total Result	**351**	**261**	**612**

Figure 245: Develop an Ad-hoc Analysis

12. Close *Analysis for Microsoft Office*. You do not need to save your workbook.

Unit 4 Solution 21

Consume SAP HANA Live Views

Business Example

As a Business user, you want to use Analysis Edition for Office in order to consume an SAP HANA Live view. This view displays sales orders that have not yet been fully processed, and allows you to aggregate the view to sum the total value of money outstanding by various analysis dimensions.

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.

1. Start Analysis for Microsoft Excel from the Windows Start Screen.
2. From the toolbar, select the *Analysis* tab, and from the ribbon, choose *Insert* → *Select Data Source*.

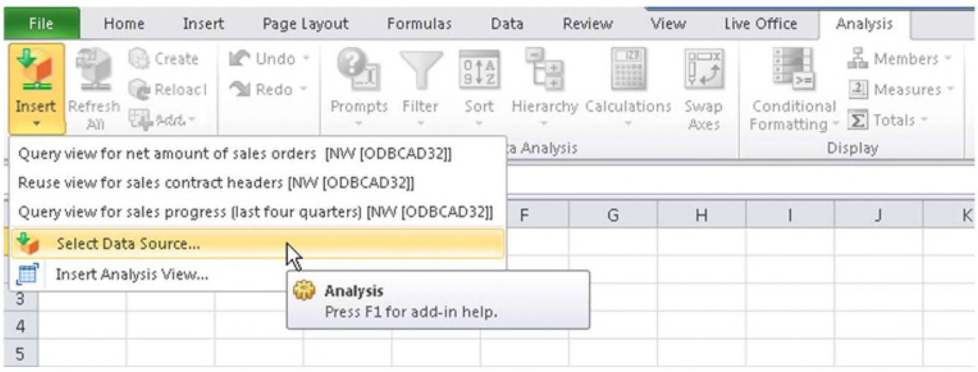


Figure 242: Data Source Analysis

> Note:
> If a *Logon to SAP BusinessObjects BI Platform* screen opens, choose *Skip*.

3. In the *Select Data Source* screen, show the connections from the Local System, choose the *HANA_##* data source and choose *Next*.

> Note:
> If another data source is already selected, choose *Back* to check the selected data source.

4. If the *HANA_##* connection does not exist, create it as follows:

- Right-click anywhere in the connections list and from the context menu, choose *Create new SAP HANA Connection*.
- Enter the following details:

Field	Value
Description	**HANA_##**
Scheme	*HTTP*
Host Name	**http://wdflbmt7215.wdf.sap.corp**
Port	**8000**
Authentication type	*Basic*

- Choose *Create and Logon*.

5. Enter the logon information from the table below and choose *OK*.

Field	Value
User	**STUDENT##** (## is your group number)
Password	**Training1**

6. Select the view type *Cube*, and in the *Search For* field, enter **opensales** and press Enter.
7. Select the *Query view for open sales orders* data source and choose *OK*.

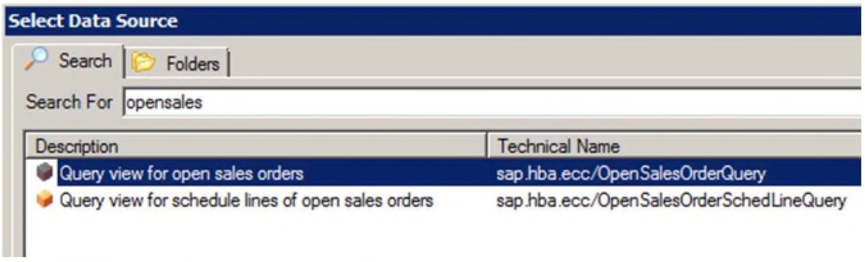


Figure 243: Locate the SAP HANA Live View

8. At the variable prompt for SAP Client enter **800** and choose *OK*.

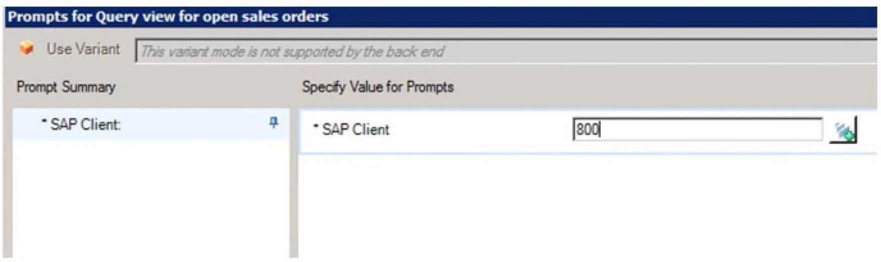
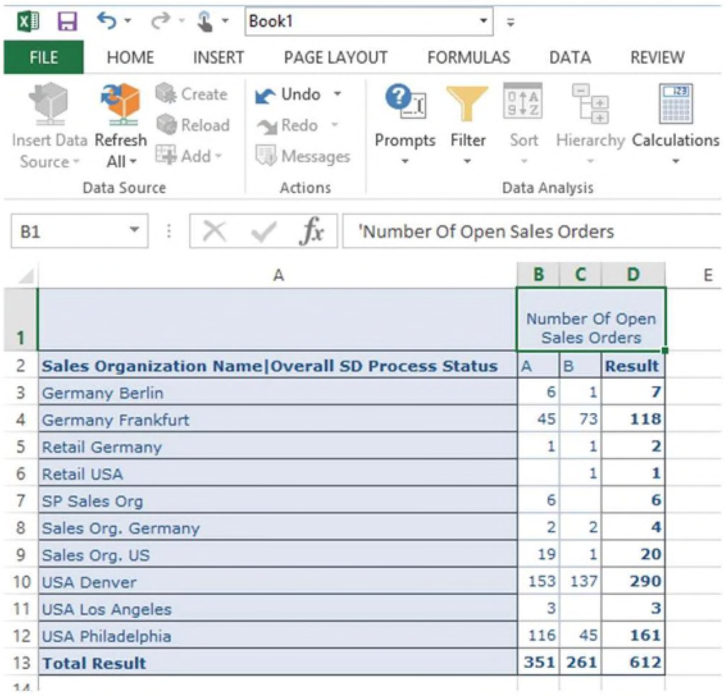


Figure 244: Complete the Prompt Entry

9. Drag the *Sales Organization Name* field to the *Rows* pane.
10. Restrict the list of measures to show only *Number of Open Sales Orders*.

    Right-click *Measures* in the *COLUMNS* pane, choose *Filter Members* and deselect the unwanted measure.

11. Add the attribute *Overall SD Process Status* to the *COLUMNS* pane.



	Number Of Open Sales Orders		
Sales Organization Name|Overall SD Process Status	A	B	Result
Germany Berlin	6	1	7
Germany Frankfurt	45	73	118
Retail Germany	1	1	2
Retail USA		1	1
SP Sales Org	6		6
Sales Org. Germany	2	2	4
Sales Org. US	19	1	20
USA Denver	153	137	290
USA Los Angeles	3		3
USA Philadelphia	116	45	161
Total Result	351	261	612

Figure 245: Develop an Ad-hoc Analysis

12. Close *Analysis for Microsoft Office*. You do not need to save your workbook.

LESSON SUMMARY

You should now be able to:

- Discover and consume SAP HANA Live views

Unit 4
Lesson 4

Modifying SAP HANA Live Models

LESSON OVERVIEW

An SAP Suite application delivers a huge number of attributes to SAP HANA Live views, but SAP have exposed only the most popular attributes for consumption. In this lesson, you will learn how to add and remove attributes using standard modeling tools and also with the SAP HANA Live Extension Assistant.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Modify SAP HANA Live Views
- Use the SAP HANA Live Extension Assistant

Overview

- You could modify **any** SAP HANA Live model
- Modify and test using **standard** SAP HANA tools in SAP HANA Studio (modeler, debug, trace)
- Detailed SAP HANA modelling **knowledge** is required
- **Non-disruptive** to the use of standard delivered HANA Live models
- New tool available for easy view extensions
- If you will be referring to new tables, make sure they are replicated if using a side by-side scenario

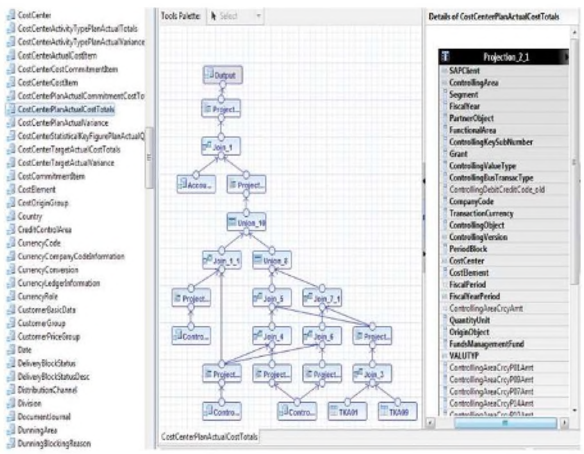


Figure 246: Modifying SAP HANA Live views — Outline

As SAP HANA Live views are technically no different from other views it is possible modify them using the standard SAP HANA modeling tools and techniques.

No special knowledge is required, just apply your standard SAP HANA modeling knowledge.

You can use all standard modeling, testing, tracing, and debug tools with SAP HANA Live views.

Examples of Modifications to SAP HANA Live views

- Remove / add user prompts to the standard models
- Rename attributes to use business user terminology
- Apply custom filters to reduce the data volume (security + performance)
- Standard models may be missing attributes from underlying Suite tables
- Standard models may expose too many attributes from Suite tables
- Add additional calculated attributes to the standard models
- Build custom hierarchies and combine with standard models
- Combine custom tables / models to the standard models
- *…..and many more reasons*

Figure 247: Examples of Modifications to SAP HANA Live Views

SAP HANA Live views do not expose all attributes and columns, as there are too many. Only the most important attributes and measures were chosen by SAP, but as SAP HANA Live views are technically no different from other views it is possible modify them using the standard modeling tools and techniques.

SAP did not know where to apply fixed filters to suit customer requirements. So you should consider adding filters to improve performance and business meaning.

SAP did not know where to apply all variables to suit customer requirements so, again, you should develop variables and add them to the model where necessary. SAP only applied filters to give the view basic meaning (for example, filter order processing status to ‘A’ and ‘B’ for Open Sales Orders). You may have added customizations to the source system, which means there are additional status codes that might need to be considered.

Consider renaming attributes and measure to suit your business requirements.

It is possible to combine custom tables and views with SAP HANA Live views.

Guidelines

- Making changes to Private, Re-Use and Query Views is not recommended
- Changes will be lost when a new VDM release is installed
- Unlike SAP BW, when you install the new content, it will **not** prompt you to use your content or the new standard version
- If a change is required the customer should first copy views manually or use the supplied tool to manage view extensions (which always copies the original view)

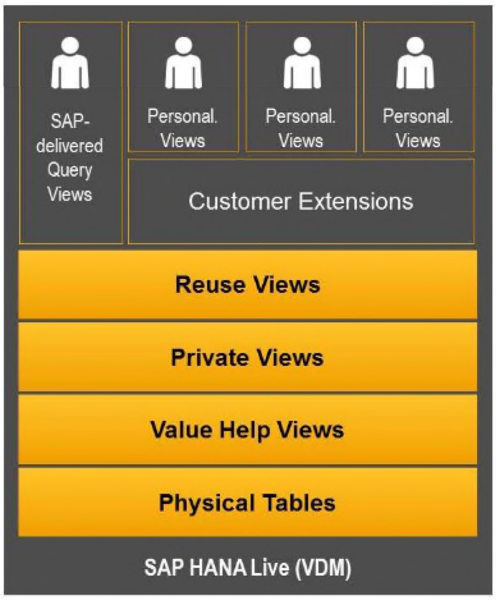


Figure 248: Modifying SAP HANA Live views — Guidelines

Do not extend SAP delivered views directly. Always work on a copy of a view in customer namespace otherwise your changes will be lost when you re-import views from new versions of the packages.

For all graphical calculation views you create, consider setting the flag, *Execute in: SQL Engine*, for best performance. This is how SAP supplied SAP HANA Live views are already set and it ensures that only the SQL engine is used. Refer to the lesson, Apply Good Modeling Approaches, to learn more about this setting.

For all enhancements, graphical calculation views should be used. If you need additional features not provided with graphical calculation views, only then should you consider script-based calculation views.

Avoid using Attribute and Analytical views unless there is a requirement to use features that are only available in these types of views. These types of views can cause the join, OLAP, and calculation engines to be invoked and when mixed with the SQL engine, this can lead to sub-optimal performance.

Hiding Attributes

1. Locate the required standard SAP HANA Live query view
2. Copy standard SAP HANA Live query view to your own namespace
3. Open the copied query view
4. Display semantics node
5. For selected attributes click 'Hidden' checkbox
6. Validate, Save and Activate the view
7. Check Data Preview to confirm attributes are now hidden
8. Build reports / applications on the new query view

Figure 249: Hiding Attributes not Removing Them

It might be tempting to completely remove an attribute or measure from the model but bear in mind that it might be referenced by other objects (for example, calculations, restrictions).

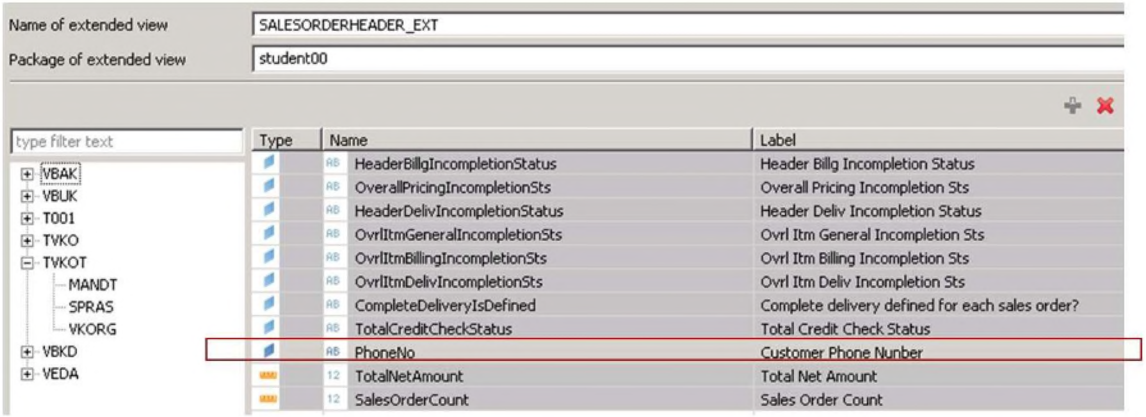
Hiding the attribute is best practice and it will ensure that it can still be referenced by other objects now and in the future, however, it will no longer be exposed to the output view for direct consumption.

Follow the steps in the figure, Hiding Attributes not Removing Them, to hide attributes or measures rather than completely remove them.

Using the Extension Assistant

The SAP HANA Live Extension Assistant is a member of the SAP HANA Live family of tools and must be installed separately. The installation steps are described in detail in the documentation at **help.sap.com**.

An easy way to determine if the tool is already installed is to check if the *Extend View* menu option appears at the bottom of the context menu for an SAP HANA Live view.



- SAP HANA Live models do not expose all standard Suite fields from ABAP tables as attributes (or measures), customers can choose to add more
- It is possible to easily achieve this using the View Extension tool delivered with SAP HANA Live

Figure 250: Extension Assistant

You can extend either a query view or a reuse view.

When extending a query view the tool automatically exposes all unused columns from all underlying reuse views. It does not expose unused columns from the source tables. You can only add columns as attributes and not measures.

When extending a query view the tool automatically exposes all unused columns from all underlying reuse views. It does not expose unused columns from the source tables. You can only add columns as attributes and not measures.

The tool also ensures that the additional fields are added to each higher level node in the model.

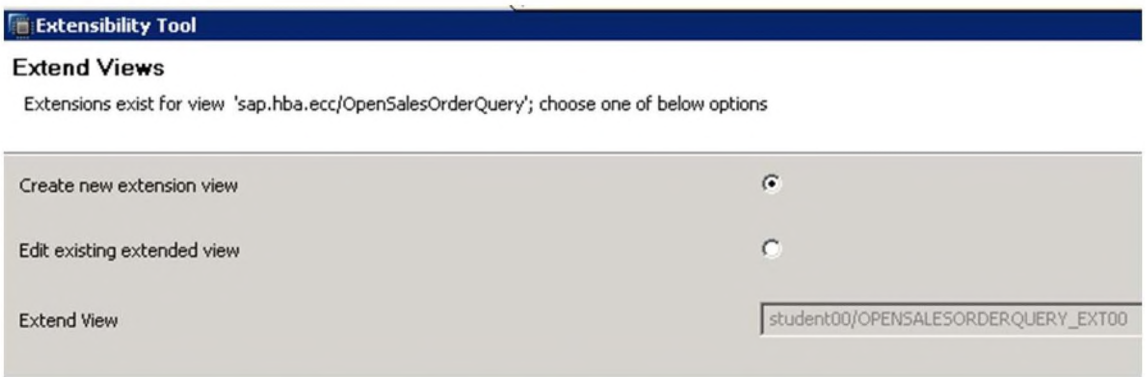
The tool cannot be used to remove standard attributes but you can use the tool to remove attributes you added earlier with this tool.

You must be assigned the role **`sap.hba.tools.extn.roles::ExtensibilityDeveloper`** to extend any view using the Extension Assistant.

Also, assign to the developer the following permissions on the packages where new extended views are created:

- REPO.READ
- REPO.EDIT NATIVE OBJECTS
- REPO.ACTIVATE NATIVE OBJECTS

New or Existing Extension



- You can extend a standard view multiple times
- Choose to edit the existing extension or create another extension

Figure 251: New or Existing Extension

You can extend a view multiple times.

If a standard view has already been extended at least once, you will see a popup and you must choose to create a new view or edit an existing one as shown in the figure, New or Existing Extension.

The Extension Assistant only works on standard SAP HANA Live views, not customer created views (the *Extend* menu option appears but it is inactive).

You cannot use the tool to extend again a generated extension view (the menu option is appears but is inactive if it is not available).

Extension Assistant Restrictions

- You cannot extend a query view that contains a union
- You cannot extend a query view that contain aggregation node in places of other than the level directly below the semantics node

Figure 252: Extension Assistant restrictions

You cannot extend query views with *Unions*. This is because the tool does not provide opportunities to update any union with the new columns added.

For such query views, the underlying reuse views can be replaced if there is a corresponding extended reuse view that takes care of this situation.

It is not possible to extend query views that have *Aggregation* nodes anywhere other than the top node (the node just below the *Semantics* node). This is because only a final aggregation on all nodes is possible. Otherwise, the measures aggregated may not make sense if you then combine nodes with the newly added fields at a higher level than the aggregation that already took place.

For such query views, underlying reuse views can be replaced if there is a corresponding reuse view.

Unit 4
Exercise 22

Extend a View using the SAP HANA Live Extension Assistant

Business Example

You want to add missing fields to a standard SAP HANA Live view. The fields customer industry and customer language have not been added in the *OpenSalesOrderQuery* view. These are important fields for your analysis and you need to add them without affecting the standard models that are already being used.

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.
>
> This activity will be performed as a simulation that you can either watch as a demo, or work through the steps manually. The steps are fully described within the simulation so you should not need to refer to the exercise and solution provided. To get started follow these instructions::
>
> 1. In Windows Explorer, open your student folder *HA300* → *HA300 Ex 22 - Extend a View Using the SAP HANA Live Extension Assistant*.
> 2. Double-click the file *index.html* to launch the simulation.
> 3. Choose either *DEMO* (just watch) or *PRACTICE* (you do the work), depending on how you want to proceed.

1. Find the query view *OpenSalesOrderQuery* and from the context menu, choose *Extend View*.

   > Note:
   > There may be a short delay before the next dialog box appears as the metadata from the model is read.

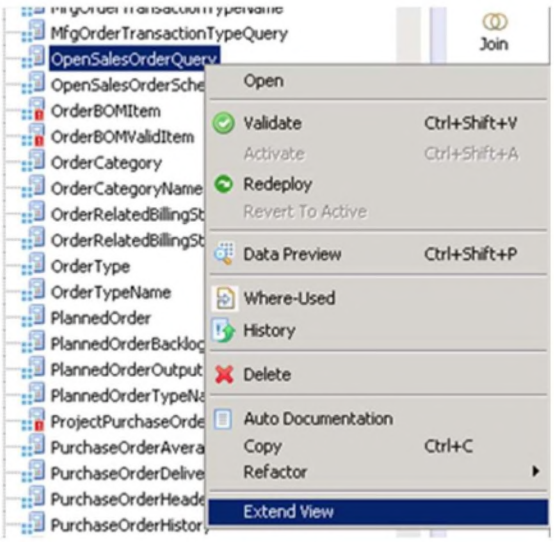


Figure 253: Choose the View to Extend

2. In the *Name of Copied View* field, enter the name **OpenSalesOrderQuery##** (## is your group number).

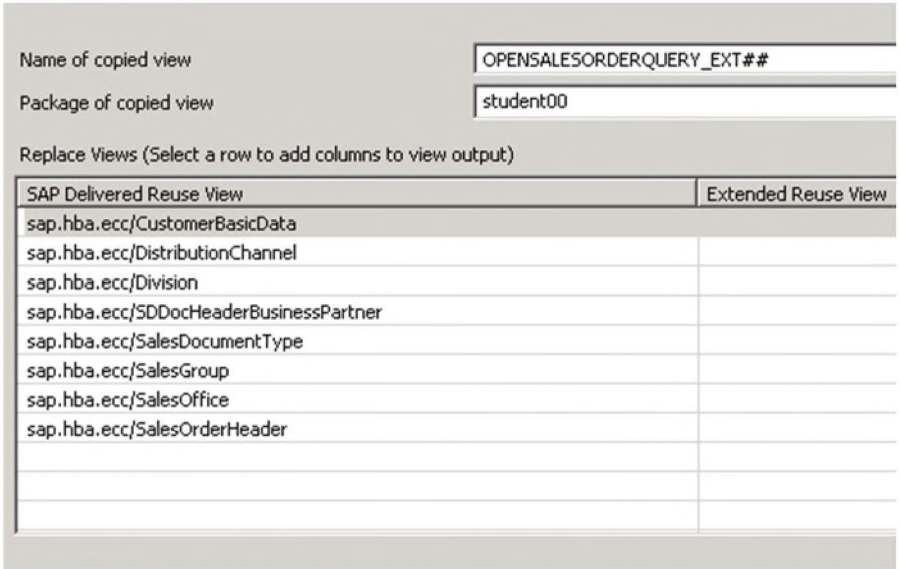


Figure 254: Name of Copied View

3. Choose the additional fields: *Language* and *Industry*. These will be added to the view.

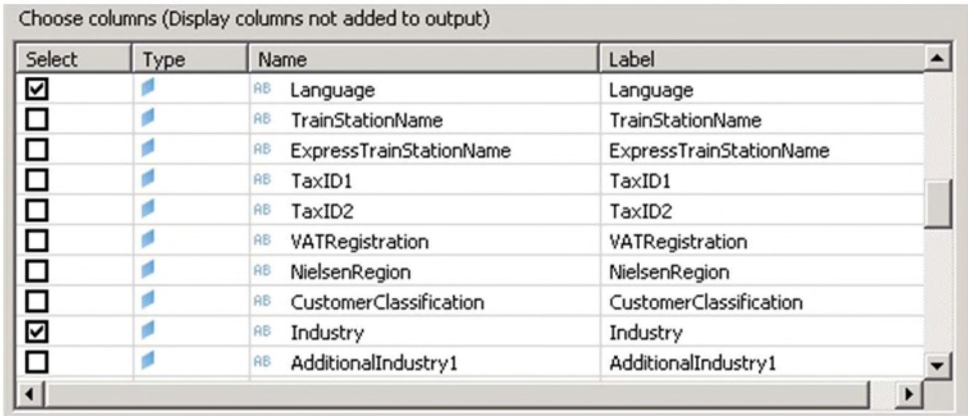


Figure 255: Select the Additional Attributes

4. Locate the newly generated query view under your package.

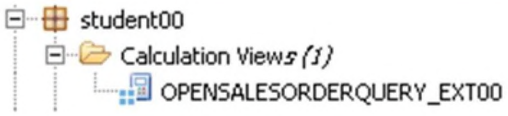


Figure 256: Locate the Generated View

5. Use the Data Preview function to view the data in the new view.

Industry	Language	NumberOfOpenSalesOrders_SUM
	D	12
	E	259
	S	3
BLD	E	18
CHEM	E	16
CON	E	44
CPNF	E	1
ELC	E	23
FOOD	D	5
HITE	D	42
	E	31
MBAU	D	31
	E	9
MFTG	E	1
MUNI	D	1
PHAR	E	13

Figure 257: Preview Data Shows the Two New Columns

Unit 4
Solution 22

Extend a View using the SAP HANA Live Extension Assistant

Business Example

You want to add missing fields to a standard SAP HANA Live view. The fields customer industry and customer language have not been added in the *OpenSalesOrderQuery* view. These are important fields for your analysis and you need to add them without affecting the standard models that are already being used.

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.
>
> This activity will be performed as a simulation that you can either watch as a demo, or work through the steps manually. The steps are fully described within the simulation so you should not need to refer to the exercise and solution provided. To get started follow these instructions::
>
> 1. In Windows Explorer, open your student folder *HA300 → HA300 Ex 22 - Extend a View Using the SAP HANA Live Extension Assistant*.
> 2. Double-click the file *index.html* to launch the simulation.
> 3. Choose either *DEMO* (just watch) or *PRACTICE* (you do the work), depending on how you want to proceed.

1. Find the query view *OpenSalesOrderQuery* and from the context menu, choose *Extend View*.

   > Note:
   > There may be a short delay before the next dialog box appears as the metadata from the model is read.

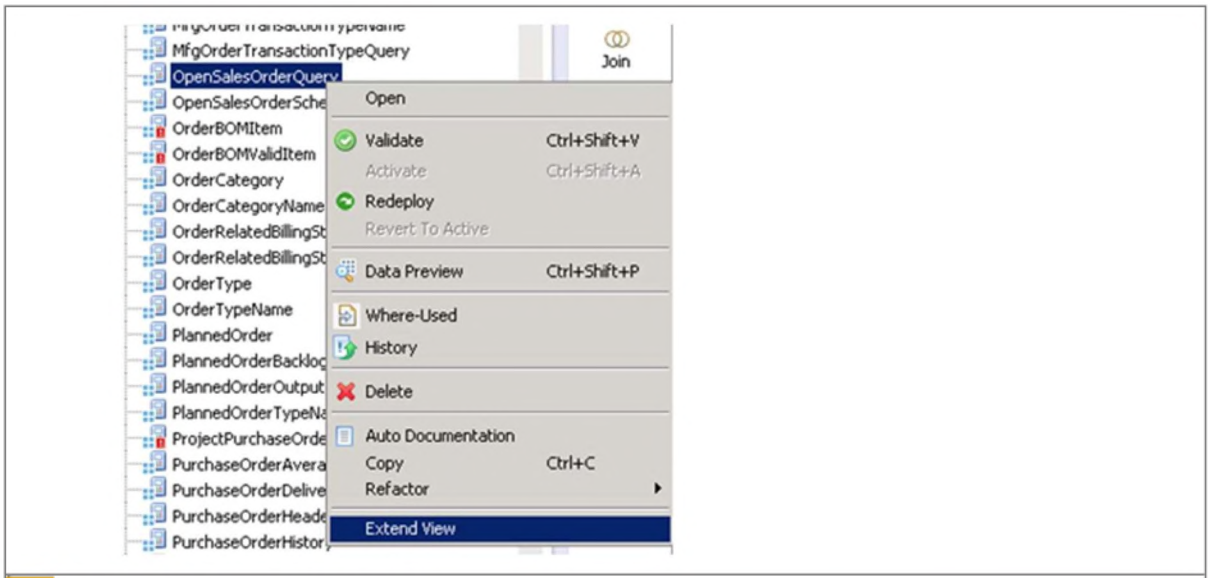


Figure 253: Choose the View to Extend

2. In the *Name of Copied View* field, enter the name **OpenSalesOrderQuery##** (## is your group number).

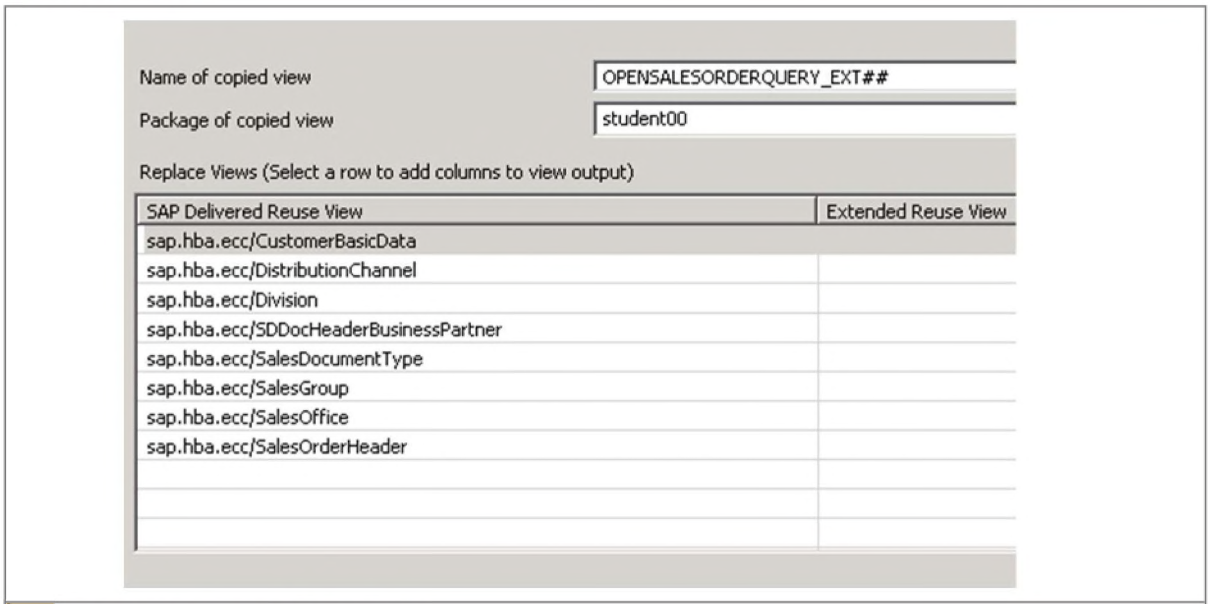


Figure 254: Name of Copied View

a) Select the *STUDENT##* package.

b) Select the reuse view *CustomerBasicData*.

3. Choose the additional fields: *Language* and *Industry*. These will be added to the view.

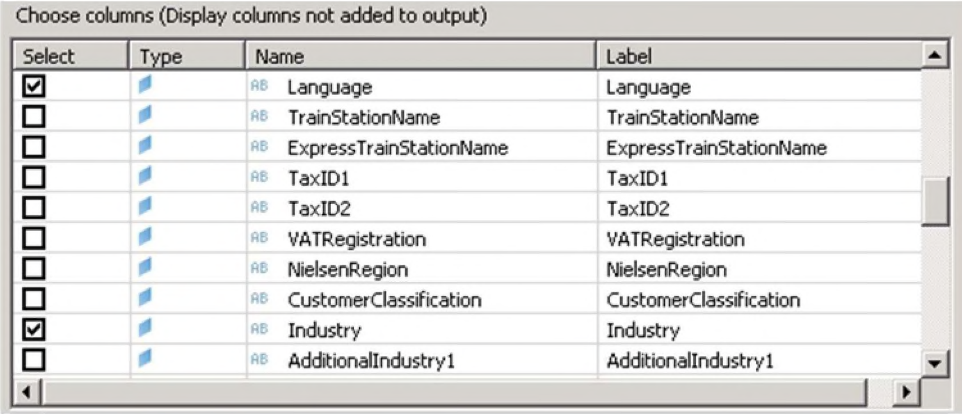


Figure 255: Select the Additional Attributes

a) Choose *Activate*.

4. Locate the newly generated query view under your package.

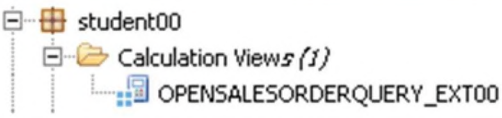


Figure 256: Locate the Generated View

a) Open the view and expand the nodes to discover where the two additional fields were added.

b) Analyze where the new fields have been added. Open the original query view to compare models. Open the original query view to compare models.

5. Use the Data Preview function to view the data in the new view.

Industry	Language	NumberOfOpenSalesOrders_SUM
	D	12
	E	259
	S	3
BLD	E	18
CHEM	E	16
CON	E	44
CPNF	E	1
ELC	E	23
FOOD	D	5
HITE	D	42
	E	31
MBAU	D	31
	E	9
MFTG	E	1
MUNI	D	1
PHAR	E	13

Figure 257: Preview Data Shows the Two New Columns

a) At the prompt, enter Client **800**.

b) Verify that the two additional fields are now part of the new query view and are ready for consumption in your report or applications.
The original view remains unaffected.

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

LESSON SUMMARY

You should now be able to:

- Modify SAP HANA Live Views
- Use the SAP HANA Live Extension Assistant

Unit 4
Lesson 5

Building a Virtual Data Model with CDS Views

LESSON OVERVIEW

In this lesson, we will introduce you to Core Data Services (CDS) views.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Leverage CDS Views to create a Virtual Data Model

Core Data Services (CDS) Views

Before application developers can begin coding, they first need to manually define all tables and views that their code will refer to. They must logon to the database and use the database tools (or SQL) to create tables and views. Only then can they create their application code. Now the application developer realizes they need to add an additional column to a table. So they go back to the database, drop the table and add the column and refill the table, and then they go back to the application code. As well as this being very cumbersome and time consuming approach, it also means the database objects are not part of the application development artifacts for the purpose of transportation to production systems. It means you would have to transport the application and then transport the database objects separately and try to ensure that these are synchronized in the target system.

What we need is a way to combine the development of the application code with the development of database objects all in one place to make it easier for developers to work with these artifacts. We also need to ensure that both application and database artifacts are kept together for transports.

The problems we described can be solved by using CDS views.

CDS View Concept

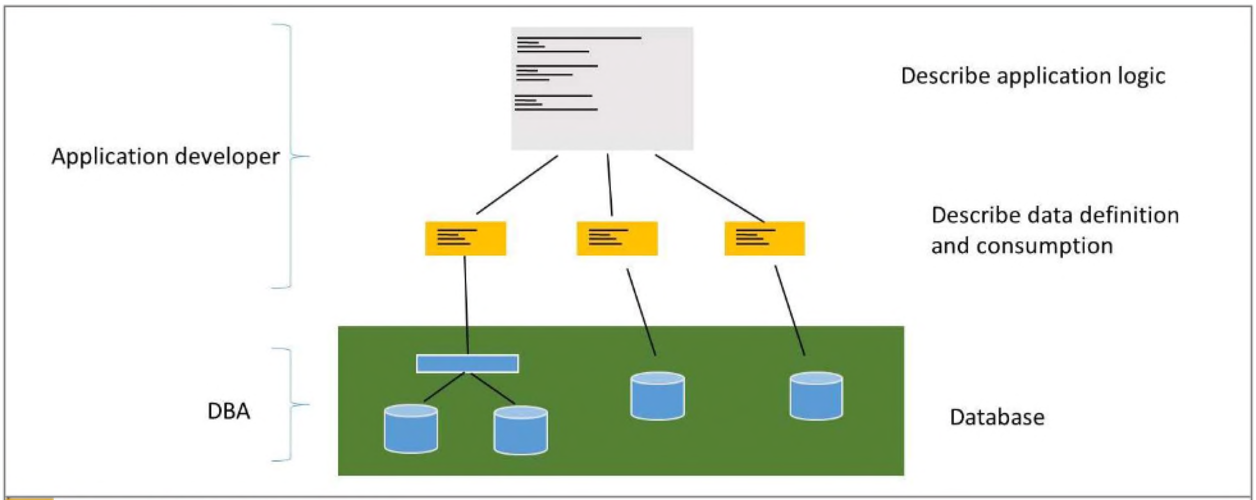


Figure 258: CDS View Concept

Core Data Services (CDS) provide a high level abstract language for defining tables and views and also how they are consumed. These definitions sit between the application code and the database. We simply describe what we need from the database using a data definition language (DDL) to create CDS views. CDS views are defined using a declarative language, so you simply state intent and do not technically describe how the data should be processed. This makes them pretty easy to work with.

CDS views are design time artifacts. So we first define the CDS view and then activate to generate the database objects, such a tables and views. It is at this point that we then also have the corresponding run time objects. When we adjust the CDS views, the database objects are automatically adjusted, even if they contain data.

CDS views can be associated with other CDS views to create a relational model. This is similar to database table joins, but the association is defined at an abstract level not a physical level. In other words, you simply declare that CDS views should work together, but you don't have to define fixed join columns. We call these associations. You can also change a CDS view once and all other CDS views that refer to it will change too. For example, if we add a column to a table definition we don't have to worry about changing all the other CDS views that refer to it.

Best practice would be to define global CDS views and then reuse them in different projects by adding the local business semantics. It might help to think of a hierarchy of layers of CDS views where each CDS view calls another to create a comprehensive meta model.

This is how we can use CDS views to build a virtual data model (VDM), in the same way as we do with SAP HANA Live that separates the semantic layer from the database objects.

Note:
SAP HANA Live is an abstract layer that sits above the physical database objects. The key difference between CDS views and SAP HANA Live is that SAP HANA Live is defined with graphical calculation views and is not combined with application artifacts, they must be transported separately. Make sure you are clear on these very important differences. Another key difference is that SAP HANA Live relies entirely on business data access control through SAP HANA Analytic Privileges, whereas CDS views can also define the data access rules and do not have to rely on Analytic Privileges.

VDM using CDS Views

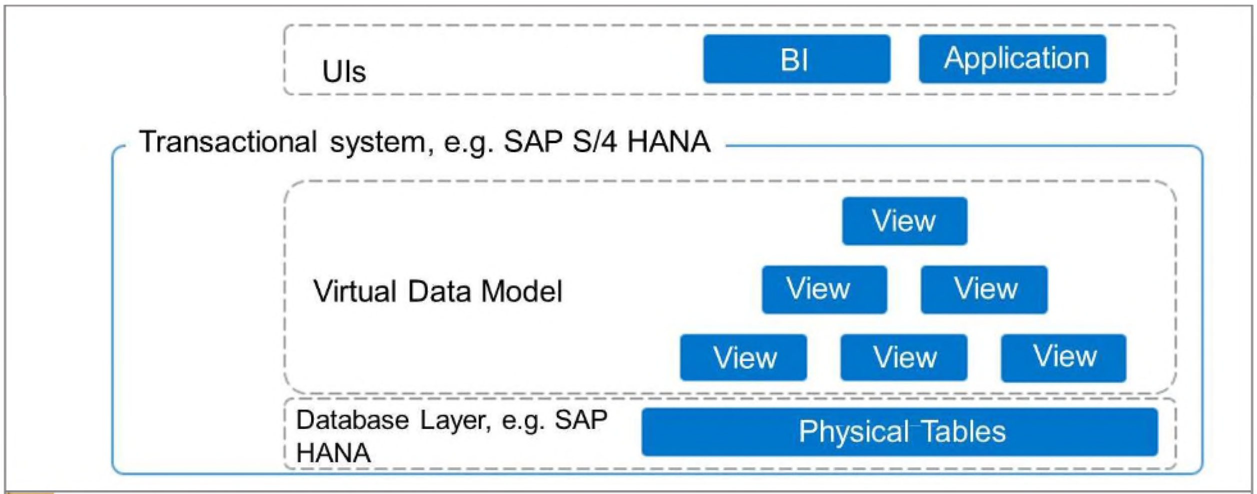


Figure 259: VDM using CDS Views

With CDS views, the developer can describe the tables and views in the application development environment using a simple language that is based on standard SQL but includes some additional syntax. CDS allows the developer not only to define the database objects needed but also to describe how the objects will be consumed. For example, we first build a CDS view to define a database view. Then we build a second CDS view to read data from the first CDS view, but we add filters, include additional calculations and aggregations that are needed by the application that is consuming the CDS view.

Although CDS views are heavily used in analytic applications that read data from SAP HANA, CDS views also play a part in transactional applications where writing to database tables is also required.

Another key aspect of CDS views is the ability to define business semantics right in the data model at a higher level that the database object. We can describe where and how the CDS views can be used. For example, a CDS view that is meant for multidimensional analysis can be described as being only allowed for OLAP reporting tools to consume. This means that the CDS view is not available to business users who require only CDS views for simple line item reporting.

Two Types of CDS View

There are two types of CDS view and it is important not mix these up when describing them. Although the concept and the design principles are the same, the actual method to define them and architecture is quite different.

- SAP HANA CDS views

SAP HANA CDS views are built and stored in a SAP HANA development repository using SAP HANA development tools. Developers who build native SAP HANA applications work with these types of CDS views. When a developer activates a CDS view in SAP HANA, the database objects, such as tables and views, are then created. Data security is managed entirely by SAP HANA using analytic privileges. When the SAP HANA application code (XS applications) is transported from one SAP HANA system to another, the CDS views are also included and both are activated on the target system together. These CDS views are native to SAP HANA, processed in SAP HANA and can only be used with SAP HANA applications.

- ABAP CDS views

ABAP CDS views are built and stored in the ABAP data dictionary of the ABAP server and not in SAP HANA. Developers who build ABAP applications work with these types of CDS views. Just as when an ABAP developer activates the definition of an ABAP table, which then generates a physical table in the database. The activation of a CDS view in the ABAP dictionary also generates a physical database object, such as a table or view. Data security is managed using the ABAP Authorisation mechanism. We can refer to existing ABAP authorisation objects and assign the business values directly in the CDS views. This is seen as a major benefit of simplification — we do not need to have two layers of security, one in the application layer and then another in the database layer. ABAP CDS views are combined with other ABAP artifacts such as application code and can be transported and activated together. These CDS views are not native to SAP HANA and can be used with other databases. The range of available SQL functions available in ABAP CDS views is similar, though not identical, to SAP HANA CDS views.

> Note:
> SAP S/4HANA uses ABAP CDS views as the entire foundation for access to data for all analytical application such as SAP Fiori applications and Smart Business Cockpits. SAP HANA Live was used heavily is early versions of SAP S/4HANA but now this has been replaced by ABAP CDS views.

CDS View Creation

CDS views are built using a family of three similar languages each with their own specific functions and syntax:

- DDL — Define tables and views and other database objects with rich semantic descriptions.
- QL — Define how to consume data from the database using filters, aggregations, calculations and other business rules.
- DCL- Define the security around data access again based on business rules.

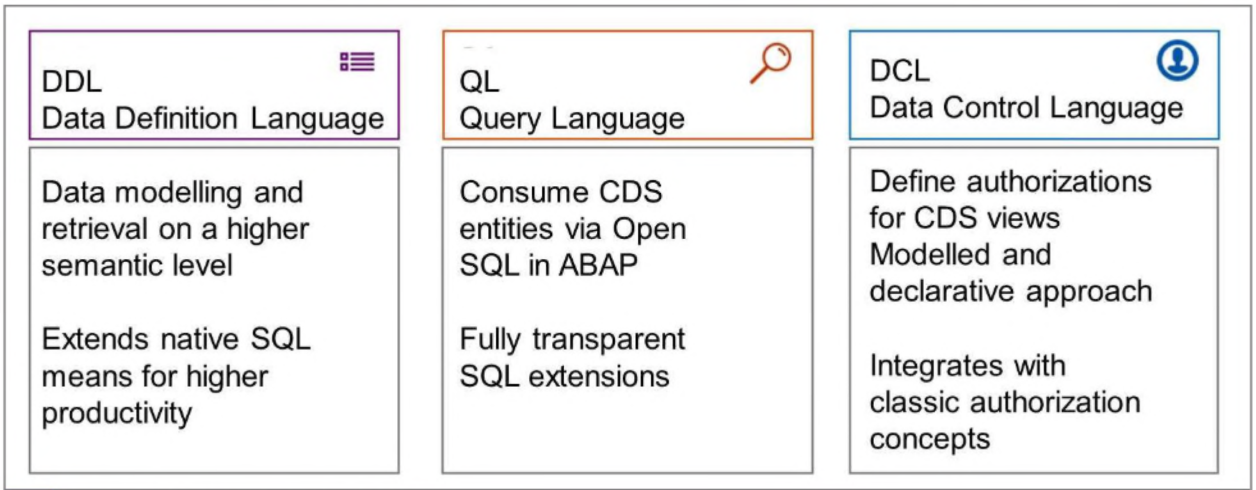


Figure 260: Languages of CDS Views

ABAP CDS views are created in the ABAP Workbench perspective of the Eclipse editor. SAP HANA CDS views are created using the SAP HANA Studio or the SAP HANA Web Workbench.

Example of Adding Semantics for Different Domains

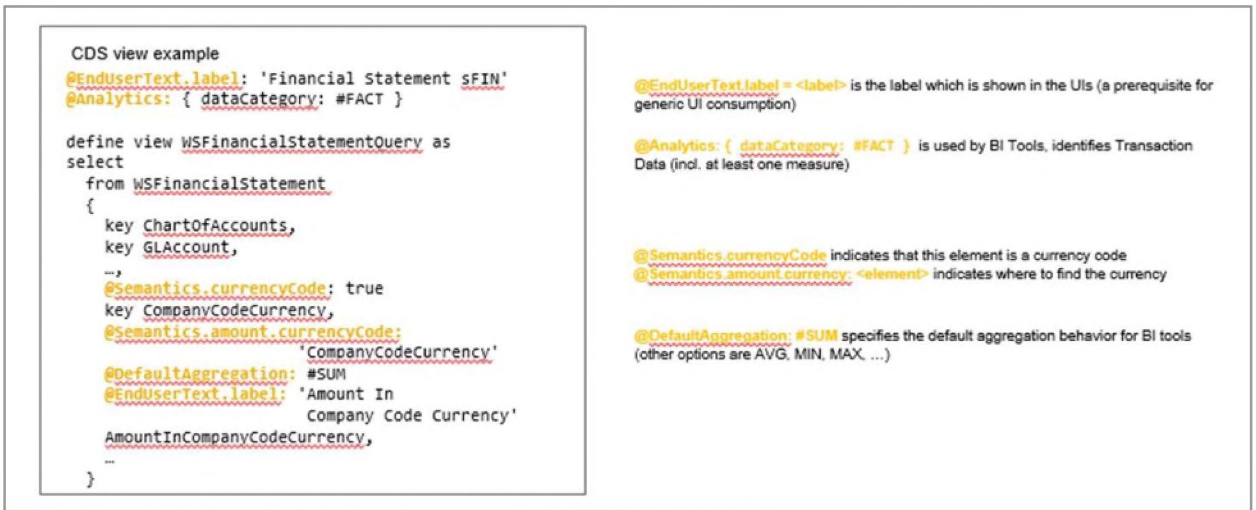


Figure 261: Example of Adding Semantics for Different Domains

The figure, Example of Adding Semantics for Different Domains, shows a CDS view that a developer has created for use by business users in their reporting tools. It has been classified as a fact table so the reporting tool knows how to handle this. Its purpose is to sum the GL amounts and present them in the currency of the company. Notice how a label 'Amount in Company Code Currency' has been defined so the consumer (business user) knows what this CDS view is meant for. This CDS view is almost certainly reading from another CDS view called 'WSFinancialStatement'. We would have to open that CDS view to see how the data is sourced.

Notice the use of the @ symbol in the CDS view? These are called annotations and are basically used to define the semantics. There is a huge number of different annotations you can add to the CDS view. Some annotations relate to security, filters, aggregation, and types of consumers allowed.

CDS Views and Native SAP HANA Modeling

SAP HANA modeling is the use of Information Views, such as attribute, analytic, and calculation views to build data models. We build Information Views mostly for BI use cases (dashboards, and operation live reports) Unlike CDS views, Information Views cannot define the database objects such as tables and views, they only describe how data is read. Using CDS views, it is possible to develop similar information models and at a basic level the outcome is the same. But not all features and functions available with Information Views are available with CDS views and vice versa. Native SAP HANA modeling is still very much needed for pure BI cases. Also, CDS views are created using a programming language and that is not the usual domain of BI modelers who are more familiar with graphical modeling tools. And finally, it is important to think about the difference in architecture and ownership. Information Views are tightly integrated in the SAP HANA database and are the responsibility of modelers. Whereas CDS views, whether ABAP or SAP HANA, are part of the application layer and are usually the responsibility of the application developers. It is certainly true that application developers will move away from using Information Views and will shift towards CDS views. But for the traditional modeler, Information Views are still the best way to build models.

> Note:
> CDS views are built using language similar to SQL. This is fine for experienced developers who code all day long and are happy in this environment. But for others this might be daunting. It is good to know SAP plan to release an easy to use graphical tool that can be used to create and maintain CDS views in the very near future. Traditional modelers, your time is coming.

So CDS views or native SAP HANA modeling? Which is better? It depends entirely on each use case. Both are relevant. We recommend that SAP HANA modelers should learn at least the basics of CDS views so they can identify where they are, and are not, used.

> Note:
> You can learn more about CDS views in the training course HA450.

LESSON SUMMARY

You should now be able to:

- Leverage CDS Views to create a Virtual Data Model

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

Unit 4

Learning Assessment

1. Which of the following statements about SAP HANA Live Browser are true?

   Choose the correct answers.

   - [ ] A The application runs in SAP HANA Studio
   - [ ] B You can develop personal lists of models
   - [ ] C The application leverages SAP Business Suite security
   - [ ] D You can display custom models as well as SAP HANA Live models
   - [ ] E You can display graphically the dependencies between views

2. Why do we use CDS views?

   ____

   ____

   ____

3. What are the 3 languages used to define CDS views?

   Choose the correct answers.

   - [ ] A Query Language
   - [ ] B Data Definition Language
   - [ ] C L Language
   - [ ] D Data Control Language
   - [ ] E R Language

Learning Assessment - Answers

1. Which of the following statements about SAP HANA Live Browser are true?

   Choose the correct answers.

   - [ ] A The application runs in SAP HANA Studio
   - [x] B You can develop personal lists of models
   - [ ] C The application leverages SAP Business Suite security
   - [x] D You can display custom models as well as SAP HANA Live models
   - [x] E You can display graphically the dependencies between views

2. Why do we use CDS views?

   To define a semantically rich data model layer that sits above the database and is used to define and consume all data

3. What are the 3 languages used to define CDS views?

   Choose the correct answers.

   - [x] A Query Language
   - [x] B Data Definition Language
   - [ ] C L Language
   - [x] D Data Control Language
   - [ ] E R Language

UNIT 5 Additional Modeling Possibilities

UNIT OBJECTIVES

- Explain the Full Text Search capabilities of SAP HANA
- Invoke the text search processes
- Create a Fuzzy Search
- Explain Text Analysis
- Define full text indexes
- Explain which data types can be indexed for Full Text Search
- Enable columns for Full Text Search in SAP HANA tables

- Use Full Text Search
- Use Fuzzy Search
- Explain Fuzzy Search relevance scoring
- Use Freestyle Search
- Work with Geospatial Data
- Query geographical data using SQL
- Develop a predictive model using SAP HANA Predictive Analysis Library
- Describe graph processing with SAP HANA

Unit 5
Lesson 1

Implementing Full Text Search and Text Analysis

LESSON OVERVIEW

This lesson is an introduction to the text search and text analysis of SAP HANA.

Business Example

In your corporate memory, you have a lot of unstructured data in text format. Data quality is also an issue within your business.

You have decided to make use of the SAP HANA Full Text Search and Text Analysis features to explore this big amount of data, in a more effective way than with traditional text processing approaches.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Explain the Full Text Search capabilities of SAP HANA
- Invoke the text search processes
- Create a Fuzzy Search
- Explain Text Analysis

Introduction to Full Text Search

A typical challenge in business today is to search for information among huge amounts of data, constantly collected in structured and unstructured forms in different areas in or outside your company. You may need then to extract meaningful information out of your data, and be able to analyze it properly.

For example, you may need to perform fault-tolerant searches, that is, when looking for duplicate master data where mistyped names are common. You may need to perform linguistic searches, that is, you search for the string **`computed`** and you would like to see also **`compute`**, **`computing`**, and **`computes`** in your search results.

You may also have additional search requirements, like ranking your search results or search one string at once over all the fields of a table. SAP HANA offers native search capabilities to support these and other business needs with the Full Text Search.

What types of text processing capabilities are supported?

Search

In addition to string matching, HANA features **full-text search** which works on content stored in tables or exposed via views. Just like searching on the Internet, full-text search finds terms irrespective of the sequence of characters and words.

Text Analysis

Capabilities range from basic tokenization and stemming to more complex semantic analysis in the form of **entity and fact extraction**. Text analysis applies within individual documents and is the foundation for both full-text search and text mining.

Text Mining

Text mining makes semantic determinations about the overall content of documents relative to other documents. Capabilities include **key term identification and document categorization**. Text mining is complementary to text analysis.

Figure 262: Full Text Search

Fuzzy Search

Among the search capabilities in SAP HANA, the Fuzzy Search offers the possibility to search for approximate matches to a given string.

For example, you could search for the string **`Walldorf`** and find all exact and approximate (to a degree that you can specify in your search) matches, like **`Walldorf`**, **`Wadlorf`**, **`Valldorf`**, and **`Wahldorff`**.

In general, a Fuzzy Search will look for all strings matching the given one exactly or differing by missing, added, or incorrect characters. The behavior is similar to what you experience when you search for a web page on an internet search engine. You do not want to see only exact matches of your search string.

- Fuzzy Search is a fast and fault-tolerant search feature for SAP HANA. The term ”fault-tolerant search” means that a database query returns records even if the search term (the user input) contains additional or missing characters or other types of spelling error.
- Examples of applications include:
  - **Fault-tolerant search in text columns** (for example, html or pdf): Search for documents on 'Driethanolamyn' and find all documents that contain the term 'Triethanolamine'.
  - **Fault-tolerant search in structured database content:** Search for a product called 'coffe krisp biscuit' and find 'Toffee Crisp Biscuits'.
  - **Fault-tolerant check for duplicate records:** Before creating a new customer record in a CRM system, search for similar customer records and verify that there are no duplicates already stored in the system.

Figure 263: What is Fuzzy Search?

SAP HANA Full Text Search

The Text Analysis features of SAP HANA allow the extraction of structured information from unstructured information.

For example, it enables linguistic markup, such as identifying the part of a speech (verbs, nouns, or adjectives). It is also useful to identify entities (locations, persons, or dates) in an unstructured text.

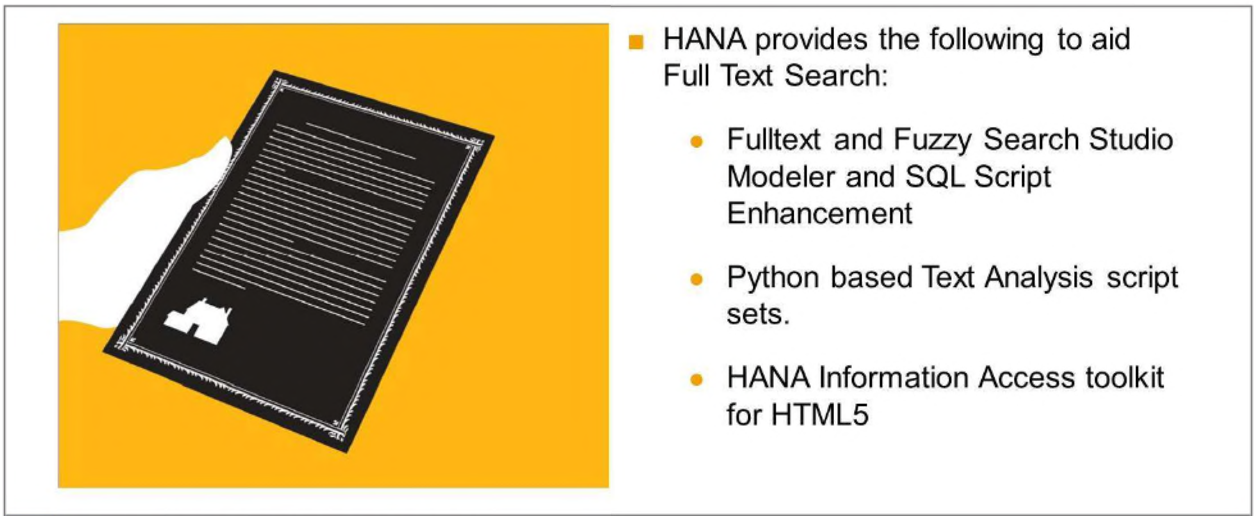


Figure 264: SAP HANA Full Text Search

Full Text Search Benefits

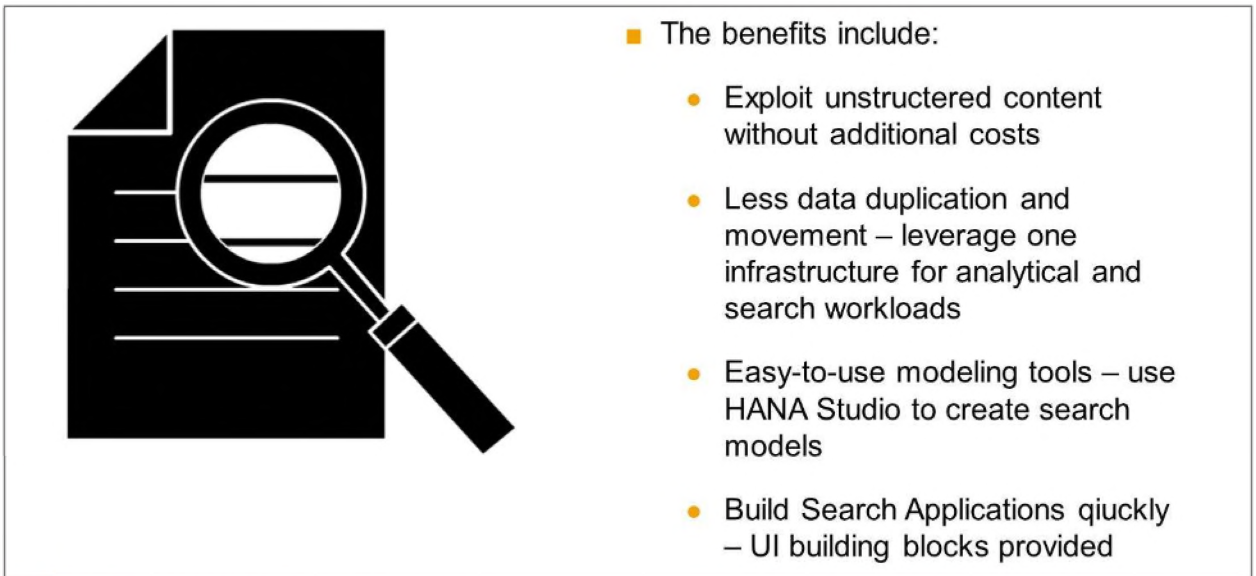


Figure 265: Full Text Search Benefits

Full Text Search Processes

- Full Text Search functions are invoked by a dedicated process

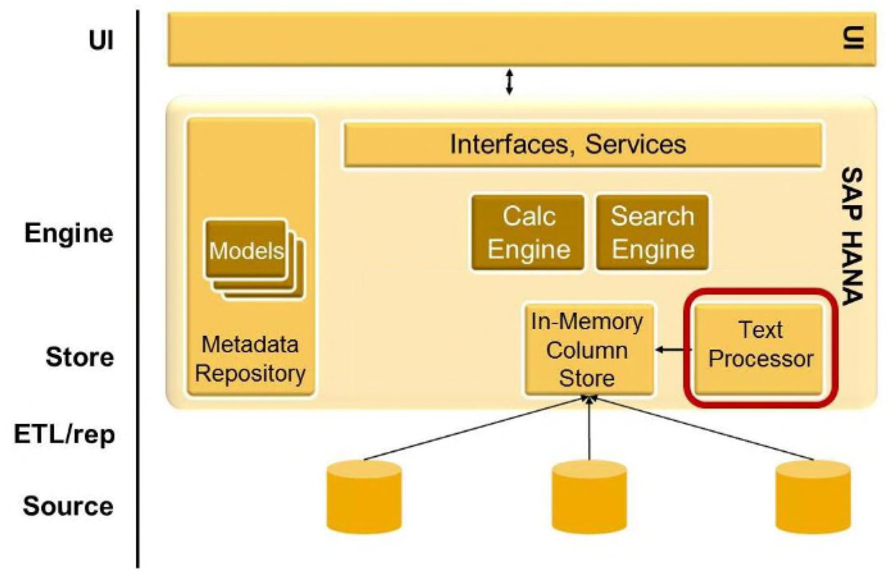


Figure 266: Full Text Search Processes

Full Text Search Processes-Models

- To enable Full Text Search, Search Models are created in HANA Studio by the modeller.

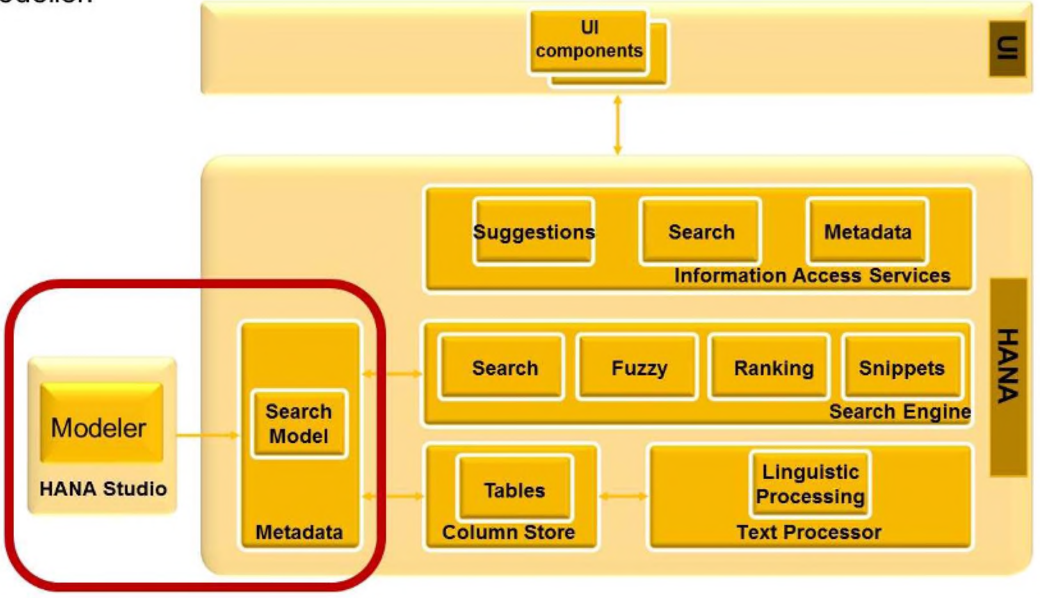


Figure 267: Full Text Search Processes

Full Text Search Processes-Information Access Services

- During data access through **Information Access Services**, Fuzzy Search queries are routed to the **Search Engine**, whilst Linguistic Processing is handled by the **Text Processor**.

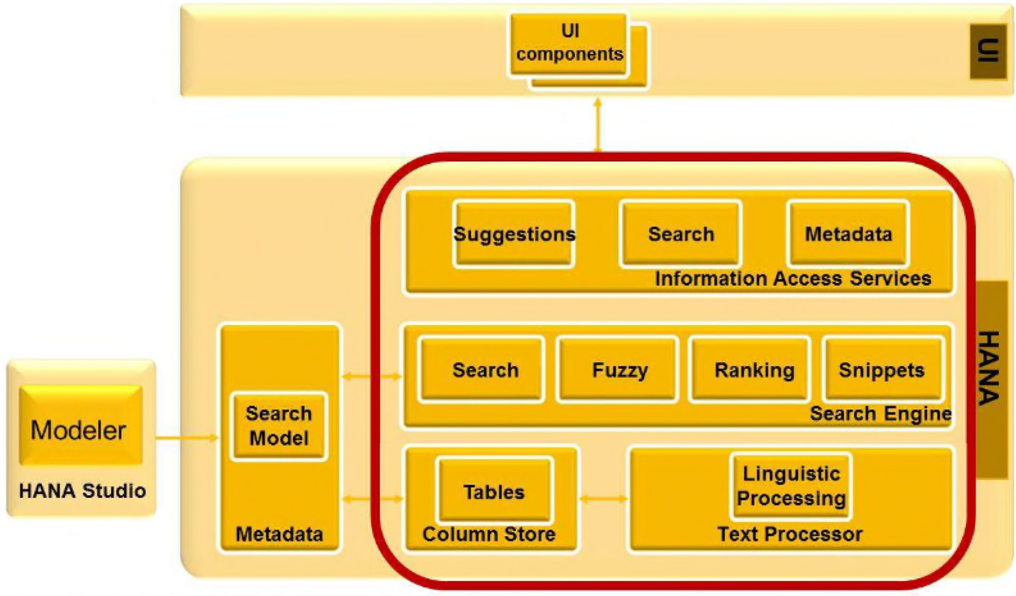


Figure 268: Full Text Search Processes

Full Text Search UI Toolkit

- The **HANA Info Access toolkit for HTML5** provides User Interface building blocks for developing search-based applications on SAP HANA. The toolkit is based on HTML5 and Javascript.

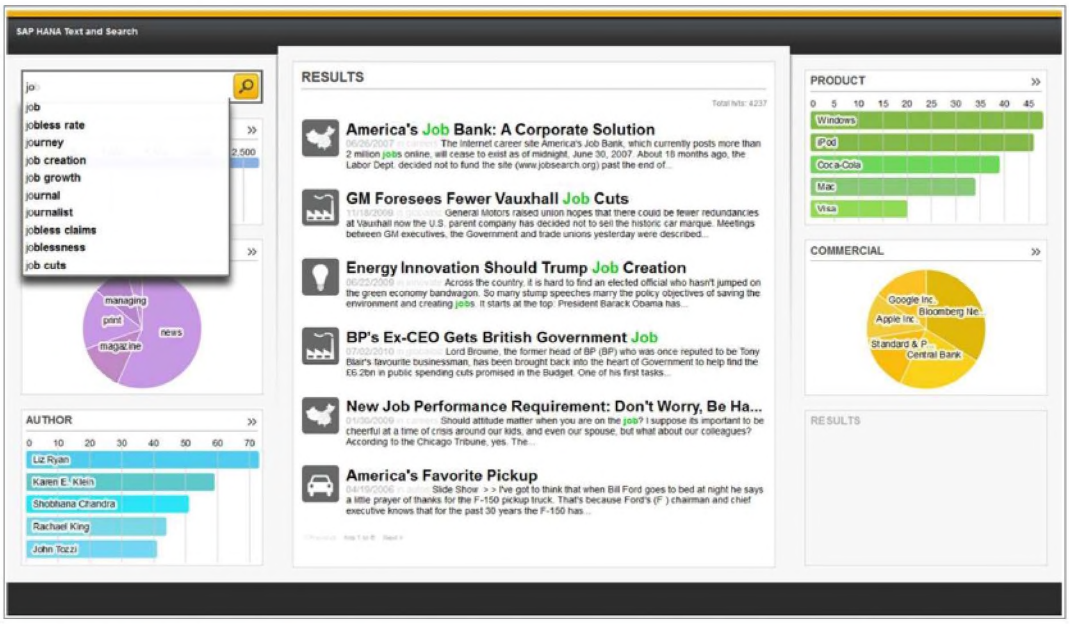


Figure 269: Full Text Search UI Toolkit

Additional Features and Components to Leverage Full-Text Search

A number of features, services or components, such as HTTP services or Application Programming Interfaces (API), enable modelers and developers to expose the Full-Text

Search results to the end user, and others are dedicated to load content (especially unstructured content) into the SAP HANA database.

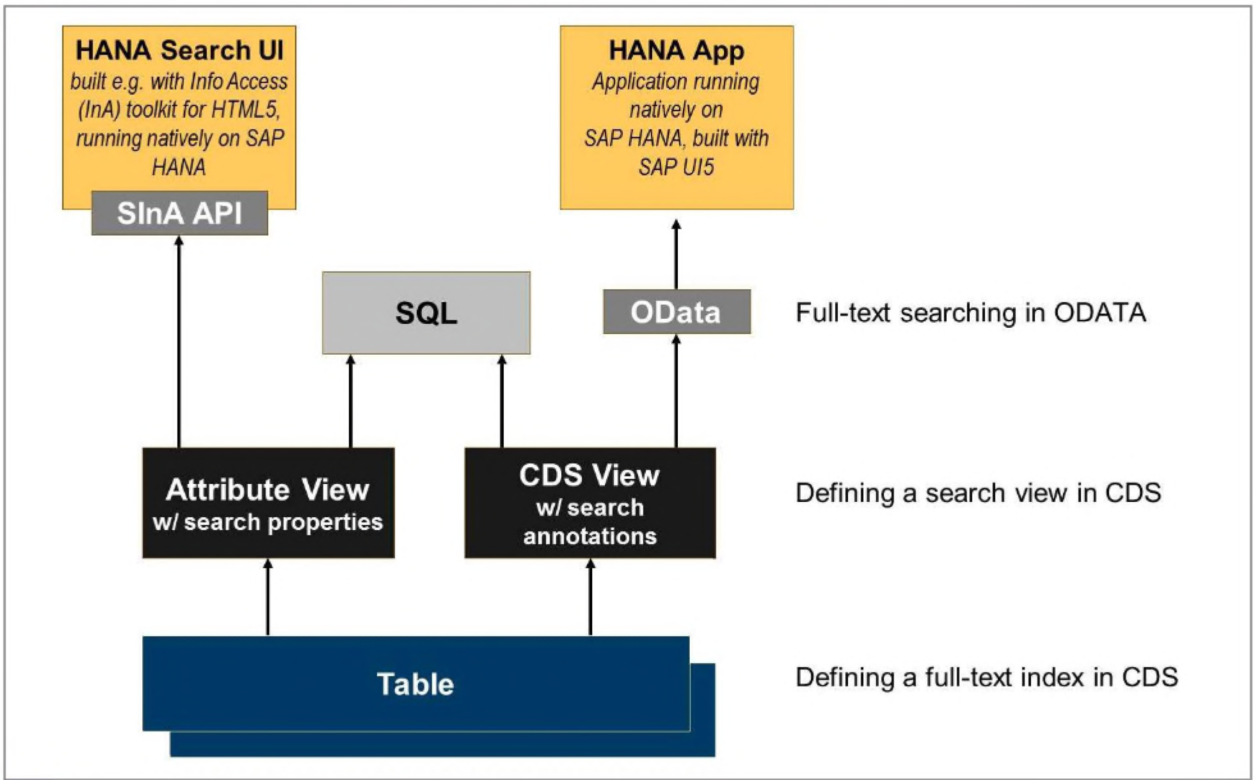


Figure 270: Full-Text Index, Search Models, and Data Access

The figure, Full-Text Index, Search Models, and Data Access, gives an overview of how Full-Text features are exposed and consumed, from the DB layer (Full Text Indexes) to the front-end tool.

The main building blocks include the following:

Core Data Services (CDS)

Core data services (CDS) is an infrastructure that can be used to define and consume semantically rich data models in SAP HANA.

CDS documents are design-time source files that are used to define the objects you want to create in the SAP HANA Catalog.

In the case of Full-Text Search, CDS documents can contain annotations (syntax) to create the following:

- Full-Text indexes
- Search models and advanced functions that you can use to consume the indexed data via ODATA

ODATA (Open Data Protocol)

From SAP HANA 1.0 SPS09, it is possible to expose search-enabled CDS views in ODATA and execute full-text searches by using ODATA.

The SAP HANA Simple Information Access (SInA) API

The SInA API is a client-side JavaScript API for developing browser-based search User Interfaces.

The toolkit is based on HTML5 and Javascript.

SAP HANA File Loader

The SAP HANA file loader is a set of HTTP services that you can use to develop your own applications to search in file contents. The file loader package also contains a basic example application with monitoring and statistical information about the current file loader schedule.

SAP HANA File Loader

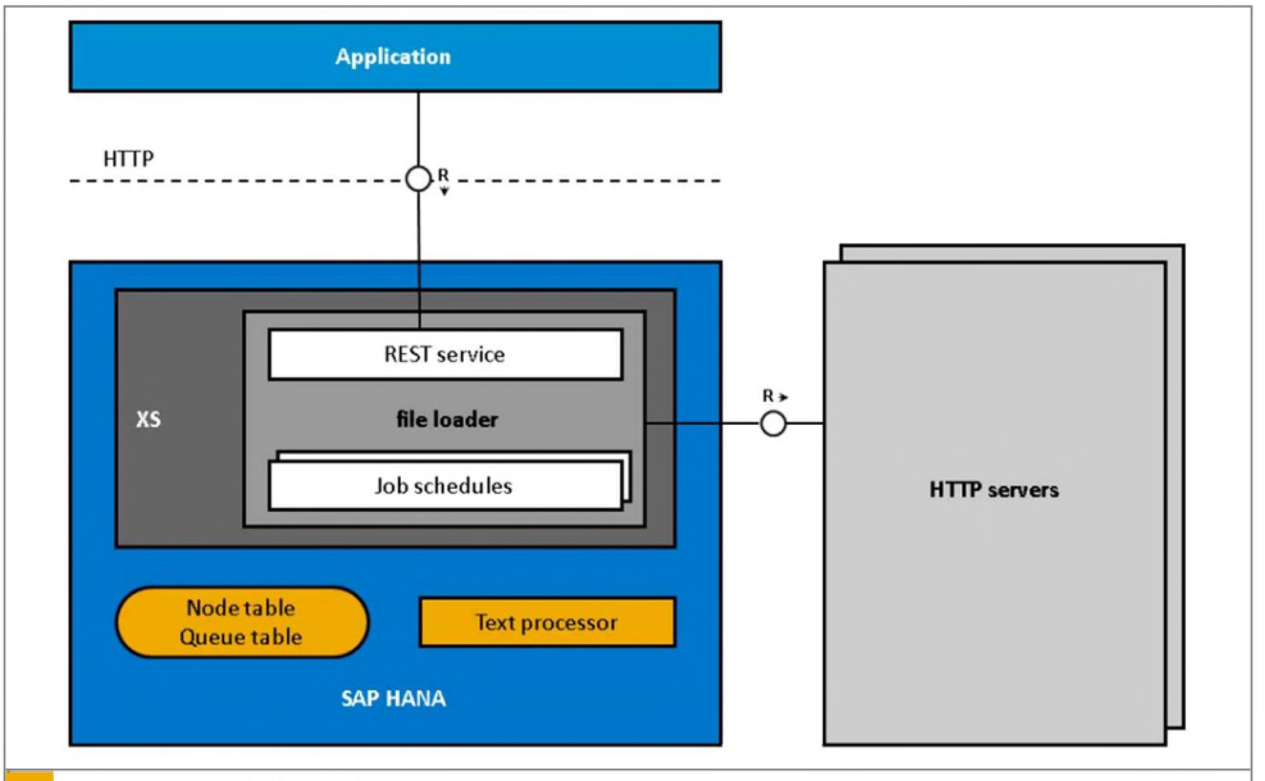


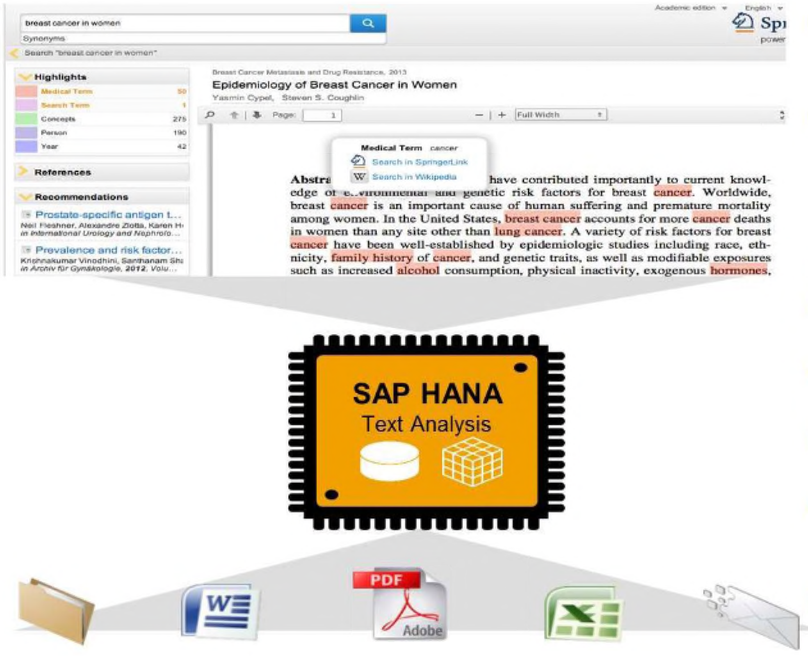
Figure 271: SAP HANA File Loader

The SAP HANA File Loader supports the loading of file contents for search. It can be used to load binary files into SAP HANA. Once this has been done, it is possible to run services like search on the content.

The component enables functionalities, such as the following:

- The support of SSL, HTTP proxies, and HTTP authentication through HTTP destinations
- A user interface dedicated to administration tasks
- Enhanced monitoring functions
- A simplified security setup and maintenance

Overview of Text Analysis



File Filtering

- Unlock text from binary documents
- Ability to extract and process unstructured text data from various file formats
- Load binary, flat, and other documents directly into SAP HANA for native text search and analysis

Native Text Analysis

- Give structure to unstructured textual content
- Expose linguistic markup for text mining uses
- Classify entities (people, companies, things, etc.)
- Identify domain facts (sentiments, topics, requests, etc.)
- Supports 31 languages for linguistic mark-up and extraction dictionary and 11 languages for predefined core extractions

Figure 272: Text Analysis

Text Analysis is a feature enabled with the Full-Text index to discover and classify entities in your documents.

Technically, it is a set of Python-based scripts.

Text Analysis provides a large number of pre-defined common entity types such as names, countries, and companies. Text Analysis also provides rules for how these entities are identified in strings. For example, if we detect the word 'Inc.' then the preceding word must be a company name. These entities and rules are available for many industries, and many languages but can easily be customized.

The language modules included with the software contain system dictionaries and provide an extensive set of predefined entity types, for example, a list of all possible country names. The extraction process can extract entities using these lists of specific entities.

Extraction classifies each entity extracted by entity type and presents this metadata in a standardized format. You can also customize the text analysis process and define your own entity types.

Individual text analysis options are grouped into text analysis configurations, which are stored in the SAP HANA repository in an XML format.

Example of Text Analysis Features

The system includes a number of predefined configurations. You can use any of these, or create your own custom text analysis configurations to apply your own text analysis extraction dictionaries and extraction rules.

Examples of standard predefined configurations include the following:

- Common entities, such as organizations, persons, countries, dates, or measures.
- Specialized extraction content, such as Marketing ("Voice of the Customer") where expected customer sentiments and feedback can be identified using the language that the customer uses.

The figure, Example of Text Analysis Features, gives two simple examples of the types of text analysis that you can perform on unstructured data.

To set up a sentiment analysis you have to create a FULLTEXT index on a column table referring specifically to the column that stores the text to be analyzed (such as a Tweet) with a predefined configuration of sentiments, such as 'Voice of Customer' that defines the possible dictionary of sentiments and whether they are positive or negative sentiments and how strong.

- **Named Entity Recognition**
  Identification of persons, companies, dates etc. in "text"

  <PERSON>John</PERSON> sold <CURRENCY>$300</CURRENCY> shares of <ORGANIZATION>Acme Corp.</ORGANIZATION> in <DATE>May 2013</DATE>

- **Sentiment Analysis**
  Extract subjective information like positive or negative sentiments, problems etc.

Figure 273: Example of Text Analysis Features

Of particular interest to most organizations is the extraction of entities and facts relating to the enterprise. Since SPS10, the entity dictionaries relating to the enterprise domain were significantly improved and also the extraction rules were enhanced.

The following major fact types are classified:

Membership Information — information about a person's affiliations
Management Changes — information about joiners, leavers, and movers
Product Releases — information about product launches, retirements, and so on
Mergers and Acquisitions — information about company mergers and acquisitions
Organizational Information — Information about the company location, structure, contacts, and so on

Grammatical Role Analysis (GRA)

Introduced with SAP HANA SPS11, we have a text analysis feature called Grammatical Role Analysis (GRA). GRA is able to deeply analyze key elements of a sentence to tag the subject, the verb, and an object. This is known as a triple. GRA is used when we need to apply deeper linguistic analysis that goes beyond single word analysis. Sometimes we need to analyze how words in a sentence actually work together in order to truly identify the context and meaning and therefore extract key information.

GRA is able to identify the role that each word plays and how each word relates to the other words in a sentence.

An Example of Grammatical Role Output

An example of the grammatical roles that are output for a sentence.

Input sentence:
Oracle was rumoured to buy marketing-software maker Responsys Inc. for $1.5 billion.

The resulting six token output is shown in the table:

Token	Dependent	Role	Governor
1	Oracle	Subject	3
2	Oracle	Subject	4
3	rumored	Root/MainVerb/Passive	(none)
4	buy	MainVerb/Active	(none)
5	marketing-software maker Responsys Inc	DirectObject	4
6	$1.5 billion	OtherObject/for	4

Figure 274: Grammatical Role Analysis

.

Text Analysis Language Support

The results of a text analysis are stored in a table and can be consumed through all supported HANA scenarios. For example, using Information Models, in R language scripts with its functions from the Predictive Analytic Library, or by using the SAP HANA Information Access toolkit for HTML5, after building a search User Interface.



Language Support

Text Analysis supports language detection of 32 languages, and identifies language to apply appropriate:

- Tokenization
- Stemming
- Part-of-Speech tagging

Available Languages	
Arabic	Indonesian
Catalan	Japanese
Chinese (Simplified)	Korean
Chinese (Traditional)	Norwegian (Bokmål)
Croatian	Norwegian (Nynorsk)
Czech	Polish
Danish	Portuguese
Dutch	Romanian
English	Russian
Farsi	Serbian
French	Slovak
German	Slovenian
Greek	Spanish
Hebrew	Swedish
Hungarian	Thai
Italian	Turkish

Figure 275: Text Analysis Language Support

Text analysis is natively integrated with all 32 languages supported by text analysis.

Introduction to Text Mining

Text mining **works at the document level**

This enables semantic determinations about the overall content of documents relative to other documents.

Functions based on Vector Space Model

- Identify similar documents
- Identify key terms of a document
- Identify related terms
- Categorize new documents based on a training corpus

Scenarios

- Highlight the key terms when viewing a patent document
- Identify similar incidents for faster problem solving
- Categorize new scientific papers along a hierarchy of topics

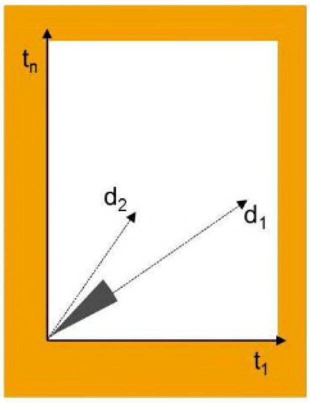


Figure 276: Text Mining Overview

Unlike Text Analysis, Text Mining performs linguistic analysis at the document level, not at the word level, and produces analysis results for each document. The basic idea is to analyze documents relative to other documents.

Imagine an oncologist wants to quickly locate all research papers of the last 12 months that discuss the use of the drug Avastin used in cancer treatments, but he specifically is interested where the side effects are being described as the main topic of the document. He wants to cluster the resulting documents by type of side-effect. With SAP HANA Text Mining you can trawl through the documents in real time to identify the documents where there is a heavy weighting towards the required subject. Text Mining uses sophisticated algorithms to produce a matrix of documents and terms.

> Note:
> Since SPS12, it is possible to interrogate the results matrix so a data scientist can carry out post-processing analysis (why did these particular documents get found?). The Term Document Matrix is a system table where the results are stored and are exposed to developers. It can be queried using SQL. The table is *$TM_MATRIX*.

Exploring Terms and Documents

Input / Output	Terms	Documents
Terms	related terms, suggested terms	relevant documents
Documents	relevant terms	related documents, categorize

Figure 277: Exploring Terms and Documents

The figure, Exploring Terms and Documents, explains the output you get depending on the input and output types (Term or Document).

Syntax Example

```
CREATE FULLTEXT INDEX myIndexName ON
myTable(myColumn)
      TEXT MINING ON;
```

Figure 278: Syntax Example

Enabling the Text Mining functionality can be done column by column. It is an additional clause to the syntax that creates the Full Text index.

Executing Queries

SAP HANA provides a server-side JavaScript interface for the Text Mining engine.

In particular, it offers the following functions:

- **getRelatedTerms**
- **getRelatdDocuments**

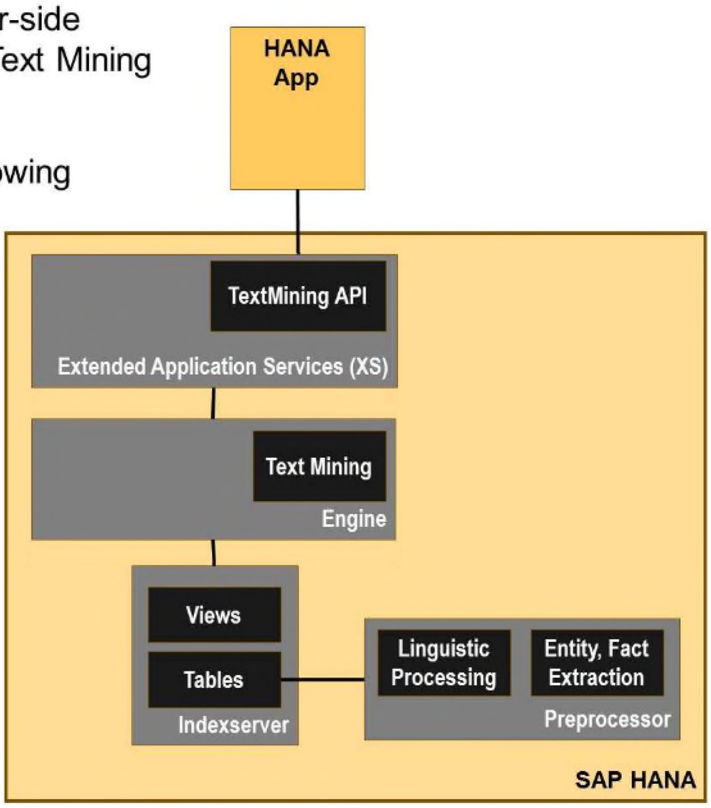


Figure 279: Executing Queries

To execute a query according to the figure, Executing Queries, data must be stored in an SAP HANA column table and a Full Text index must be defined with text mining ON.

SQL Extensions for Text Mining

Since SPS10, in addition to the XS-based Text Mining API, SAP HANA provides SQL functions to retrieve text mining statistics and enable data categorization.

Table 14: Example of Text Mining SQL Functions

Function	Retrieved Data
TM_GET_RELATED_TERMS	top-ranked terms related to a term
TM_GET_RELEVANT_TERMS	top-ranked relevant terms (key phrases) that describe a document
TM_GET_SUGGESTED_TERMS	top-ranked terms matching an initial sub-string
TM_GET_RELATED_DOCUMENTS	top-ranked documents related to a docu-ment
TM_GET_RELEVANT_DOCUMENTS	top-ranked documents relevant to a term
TM_CATEGORIZE_KNN	classification of an input document accord-ing to taxonomies (sets of categories), using the KNN (k nearest neighbor) method

```
select T.RANK, T.TERM, T.DOCUMENT_FREQUENCY, T.SCORE
  FROM TM_GET_SUGGESTED_TERMS (
    TERM 'john'
    SEARCH DISTINCT "content"
      FROM "myschema"."news"
    RETURN
      TOP 5
  ) AS T
WHERE T.SCORE > 0.1 and T.TERM_FREQUENCY > 5
```

Rank	TERM	NORMALIZED TERM	TERM TYPE	TERM FREQUENCY	DOCUMENT FREQUENCY	SCORE
1	John	john	Proper	25	11	0.86
2	John Doe	john doe	Proper	12	8	0.24
3	John Miller	john miller	Proper	5	3	0.21
4	Johnny	johnny	Proper	7	7	0.15
5	Johnson and Sons	johnson and sons	Organization	5	2	0.09

Figure 280: Text Mining SQL Functions Example

In the figure, Text Mining SQL Functions Example, the SQL code returns the top-ranked terms "matching" the text **john** from a news table.

LESSON SUMMARY

You should now be able to:

- Explain the Full Text Search capabilities of SAP HANA
- Invoke the text search processes
- Create a Fuzzy Search
- Explain Text Analysis

Unit 5
Lesson 2

Defining Full Text Indexes

LESSON OVERVIEW

This lesson will show you what data types are automatically enabled for Full Text search and what data types can be optionally configured to support Full Text Search.

You will also learn how to set up SAP HANA information models so that they support Full Text search on some of their columns.

Business Example

You have *VARCHAR* fields in your SAP HANA database, and you want to perform fast fuzzy searches on them.

For this purpose, you decide to create Full Text Search indexes on these columns.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Define full text indexes
- Explain which data types can be indexed for Full Text Search
- Enable columns for Full Text Search in SAP HANA tables

Full Text Indexes

Full Text Indexes support searching on columns in the same way that indexes support searching through books.

Unlike a string search, the sequence of words and characters used for a text search is not critical for finding matches. A Full Text Index enables this functionality by analyzing and preprocessing the available text semantically. This search includes normalization, tokenization, word stemming, and part of speech tagging.

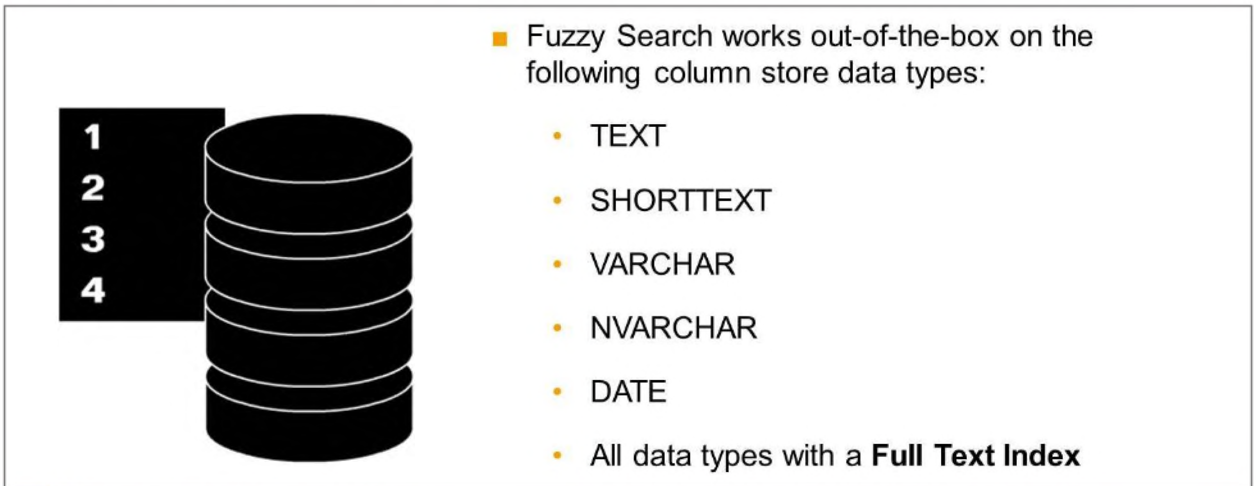


Figure 281: Supported Data Types

Creating a Full Text Index

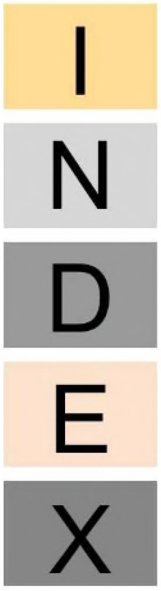
A Full Text Index is an additional data structure that is created to enable fast text searches on a specific column in a table.

Several options can be specified when creating a Full Text Index, for example:

- Which language will be used to analyze the text?
- Do you want to enable the Fuzzy Search Indexing for the column?

  Enabling Fuzzy Search Indexing will further accelerate the fuzzy searches.
- How do you want the Full Text Index to be updated? When an insert or update is performed on the underlying table: synchronously or asynchronously?

> Note:
> The complete list of available options for the creation of a Full Text Index can be found in the SAP HANA Search Developer Guide, available on the SAP Help Portal.



- It is possible to speed up the fuzzy search by creating additional data structures, which are used for faster calculation of the fuzzy score. These data structures exist in the memory only, so no additional disk space is required.
- You should enable the fast fuzzy search structures for all database columns that have a high load of fuzzy searches, and for all database columns that are used in performance-critical queries, to get the best response times possible.
- The additional data structures increase the total memory footprint of the loaded table.

Figure 282: Full Text Indexes

Full Text Indexes for Data Types

Full Text indexes are automatically created for columns with TEXT or SHORTTEXT data type, for the other supported data types you need to create them manually with a `CREATE FULLTEXT INDEX` statement.

If you want to check the parameters for an existing Full Text Index, you can query the monitoring view `SYS.FULLTEXT_INDEXES`.

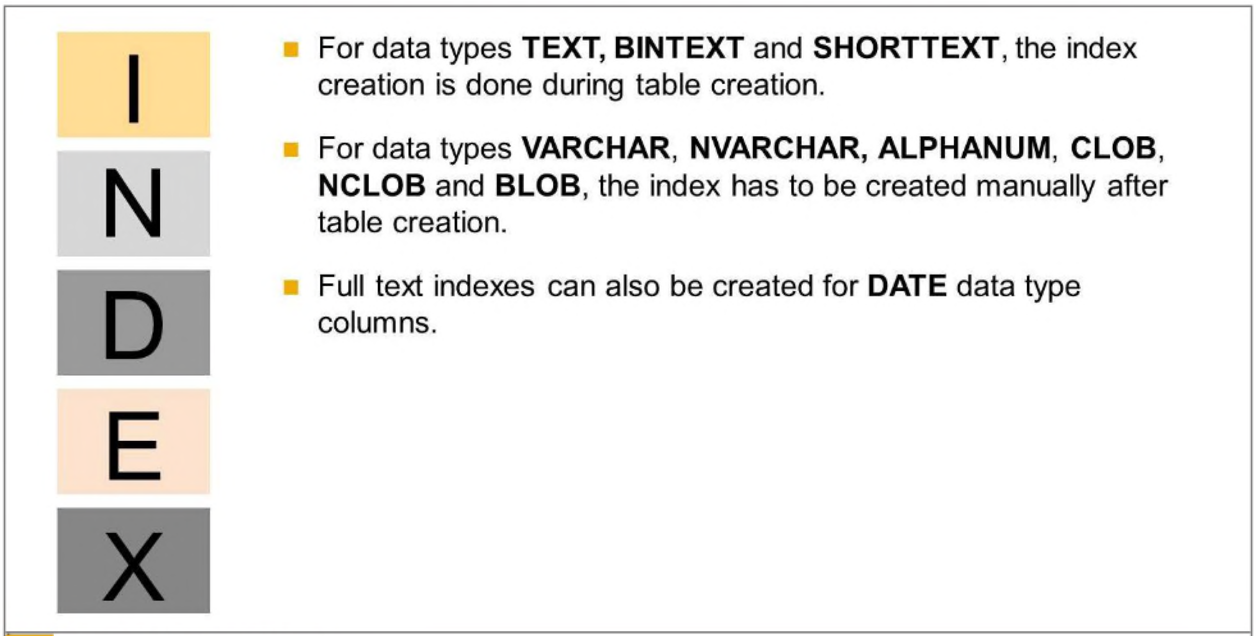


Figure 283: Full Text Indexes

Enabling a Fuzzy Search Index on TEXT or SHORTTEXT

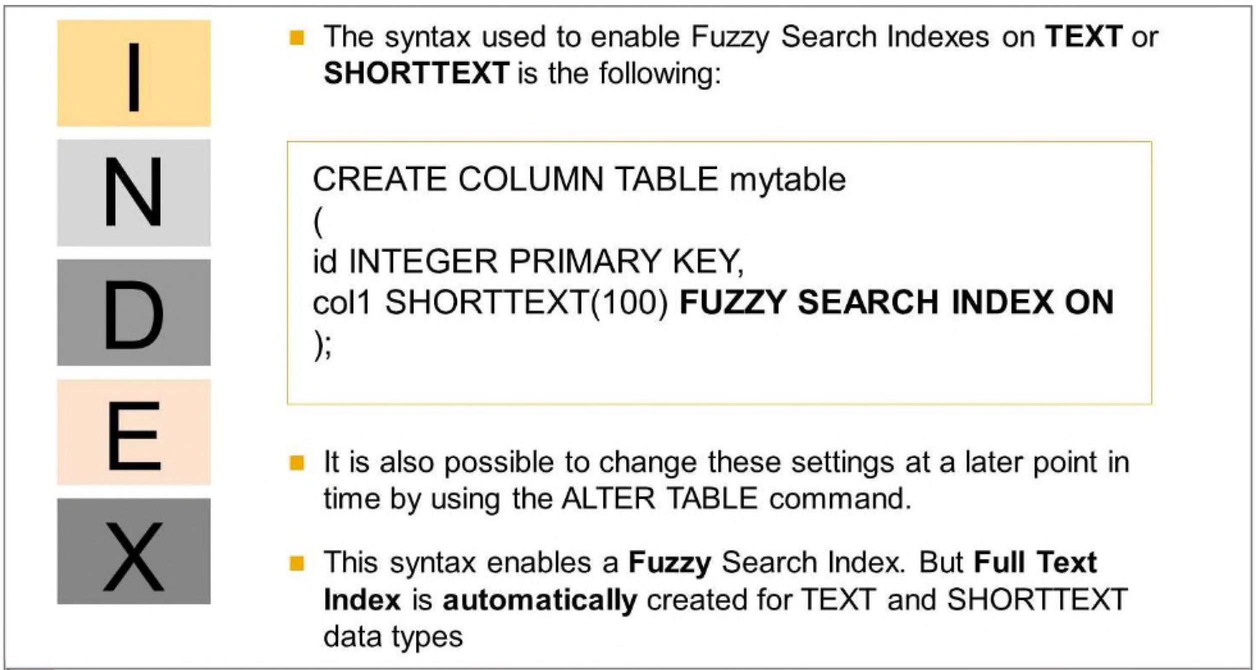
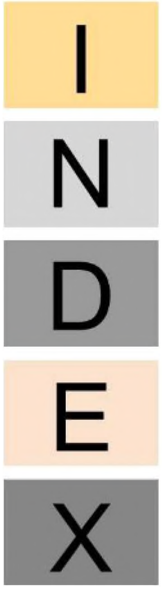


Figure 284: Enabling a Fuzzy Search Index on TEXT or SHORTTEXT

Enabling a Fuzzy Search Index on Other Data Types



- The syntax used to enable Fuzzy Search Indexes on **VARCHAR**, **NVARCHAR** or **CLOB** is the following:

```
CREATE COLUMN TABLE mytable
(
col1 NVARCHAR(2000)
);

CREATE FULLTEXT INDEX myindex ON mytable(col1)
          FUZZY SEARCH INDEX ON;
;
```

- Note that you need to create a Full Text Index, to additionally enable a Fuzzy Search Index
- This **can** be changed at a later point in time by using the ALTER FULLTEXT INDEX command.

Figure 285: Enabling a Fuzzy Search Index on Other Data Types

Monitoring the Indexing Process

```
SELECT INDEXING_STATUS("myColumn") FROM "myTable";
```

Provides information on indexing status per document

```
SELECT INDEXING_ERROR_CODE("myColumn"),
 INDEXING_ERROR_MESSAGE("myColumn") FROM "myTable";
```

Provides information on indexing errors per document

Figure 286: Monitoring the Indexing Process

SAP HANA 1.0 SPS09 brings new functions to monitor the indexing process. These functions return the indexing status (or error) for each row of a table.

Viewing Binary Content

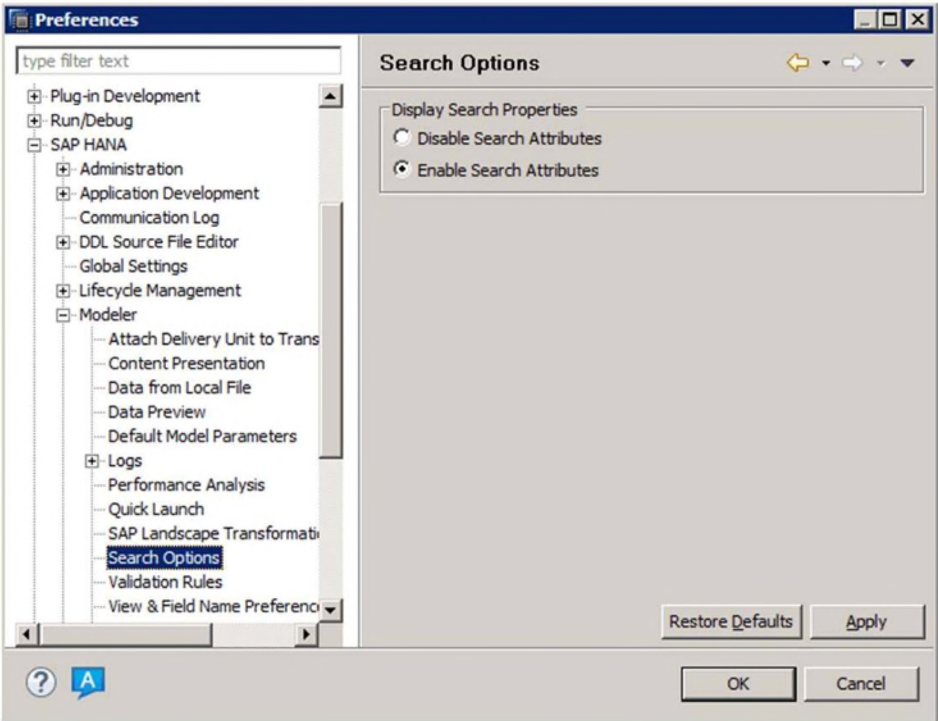
```
SELECT PLAINTEXT("myBinaryContent") FROM "myTable";
```

Figure 287: Viewing Binary Content

In case your have some binary files stored in the SAP HANA database in columns of type BLOB (Binary Large Object), it is possible to retrieve a plaintext representation of the binary file content.

Search Options in Modeling

SAP HANA → Modeler → Search Options



In order to use SAP HANA Studio for Search modeling, you need to enable **Search Options** in the preferences.

Figure 288: Enabling Search Options for Modeling

To use the *Search Options* in a view, you need to enable this functionality in the SAP HANA Studio *Preferences*.

Note:
Search attributes are not supported in graphical calculation views.

Search Properties in an Information Model

- In Attribute Views, a new tab called **Search Properties** is displayed in the properties of Attributes enabled for Full Text Search.

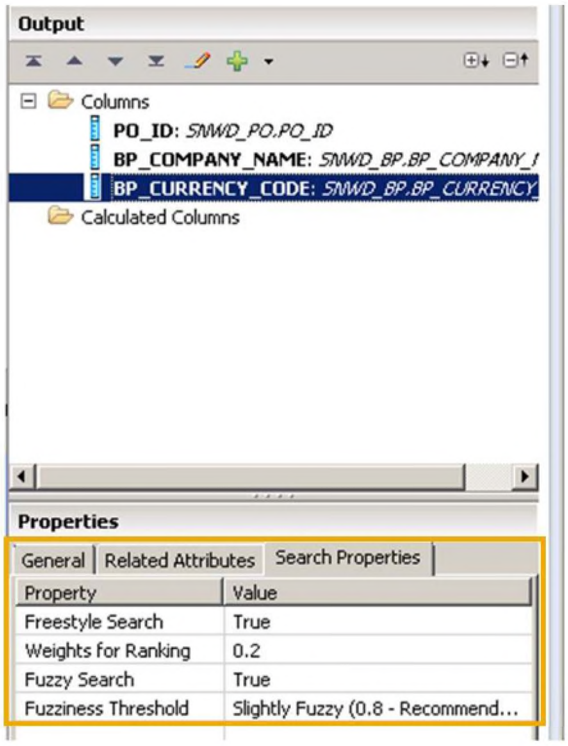


Figure 289: Search Properties in an Information Model

If the *Search Options* are enabled, you can set the *Search Parameters* for an attribute of a view. The following parameters are available:

- *Freestyle Search*

  To include or exclude an attribute in the Freestyle search.

- *Weights for Ranking*

  You can assign a specific weight to the attribute to influence the attribute relevance in the search results. Allowed values must be between *0.0* and *1.0*.

- *Fuzzy Search*

  To enable fault-tolerant searches on the attribute.

- *Fuzziness Threshold*

  Threshold for fuzzy searches. Allowed values must be between `0.0` and `1.0`.

> Note:
> If not specified, a value of 0.8 is applied by default.

Memory Impact

In SAP HANA, certain full-text index parameters can have a significant impact on memory consumption based on how they are defined.

Type	Memory Impact Details
Fuzzy Search	Fuzzy Search indexes require approximately 10% of the memory size of the column.
Text Analysis	If this parameter is set to *ON*, an additional table is created for storing structured data extracted from the source text for text analysis. The amount of extracted data depends on the data in the source column, the text analysis rules, and the structure of the results. In certain cases, the memory consumption of the extracted data could exceed the memory consumption of the source data.

LESSON SUMMARY

You should now be able to:

- Define full text indexes
- Explain which data types can be indexed for Full Text Search
- Enable columns for Full Text Search in SAP HANA tables

Unit 5
Lesson 3

Using Full Text Search

LESSON OVERVIEW

This lesson describes in detail how to use the Full Text Search features of SAP HANA, including the option for Fuzzy Search.

Business Example

You are working for a chemical company and you are searching through customer inquiries for a product called *Drieathanolamyn*.

The product name is often misspelled, and you want to make sure you do not miss any customer inquiry due to misspellings.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Use Full Text Search
- Use Fuzzy Search
- Explain Fuzzy Search relevance scoring
- Use Freestyle Search

Using Full Text Search in SQL

To perform a Full Text Search within a SQL `SELECT` statement, you can use the `CONTAINS()` function within your `WHERE` clause. If you specify a column name and a string as parameters to the `CONTAINS()`, an exact search of the string is performed.

- You call the Full Text Search by using the CONTAINS() function in the WHERE-clause of a SELECT statement.
  Without the FUZZY option the search will only return results that contain the exact phrase searched for.

```
SELECT
        SCORE() AS score, *
FROM
        documents
WHERE
        CONTAINS(doc_content, 'Driethanolamyn')
ORDER BY
        score DESC;
```

Figure 290: Full Text Search SQL Syntax

Full Text Search SQL Syntax

A first attempt in allowing fault-tolerant searches could be to search for string patterns, where you allow any character in the front and in the back of your string illustrated by the figure, Full Text Search SQL Syntax. But doing so would still not allow misspelled strings to show up in the query results.

- A fuzzy search is an alternative to a non-fault-tolerant SQL statement like the example below, which would not return any results when there are spelling errors.

```
SELECT
        ...
FROM
        documents
WHERE
        doc_content LIKE '% Driethanolamyn %' ...
```

Figure 291: Full Text Search SQL Syntax

Enabling Fuzzy Search

To perform a fault-tolerant search, you can specify a fuzziness threshold, which is a number between `0.0` and `1.0`. A threshold of *1.0* corresponds to the requirement of an exact match, the lower the threshold, the more tolerant the search. In your search results, you can also visualize a *SCORE* for each result item, where again *1.0* stands for an exact match, and *0.0* indicates it has nothing in common with the search string. Be aware that the fuzziness threshold and the *SCORE* are distinct, although related, concepts. It is possible to see a result with a *SCORE* lower than the Fuzziness threshold.

- The fuzzy search algorithm calculates a fuzzy score for each string comparison. The higher the score, the more similar the strings are. A score of 1.0 means the strings are identical. A score of 0.0 means the strings have nothing in common.
- You can request the score in the SELECT statement by using the SCORE() function. You can sort the results of a query by score in descending order to get the best records first (the best record is the record that is most similar to the user input). When a fuzzy search of multiple columns is used in a SELECT statement, the score is returned as an average of the scores of all columns used.

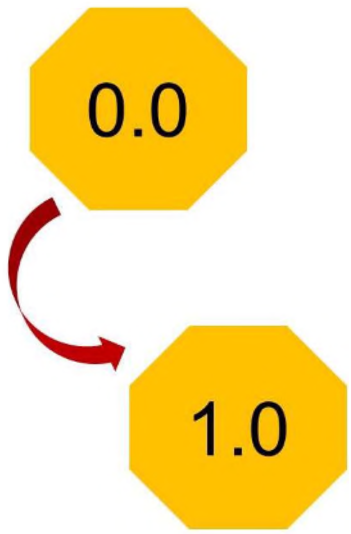


Figure 292: Fuzzy Search Relevance Score

The *SCORE* computation differs depending on the data type, as outlined in the figures, String Type Fuzzy Search, and Text Type Fuzzy Search.

String Type Fuzzy Search



String Type Search

- String types support a basic fuzzy string search. The values of a column are compared with the user input, using the fault-tolerant fuzzy string comparison.
- When working with string types, the fuzzy string comparison always compares the full strings. If searching with 'SAP', for example, a record like 'SAP Deutschland AG & Co. KG' gets a very low score, because only a very small part of the string is equal (3 of 27 characters match).

Figure 293: String Type Fuzzy Search

Text Type Fuzzy Search



Text Type Search

- Text types support a more sophisticated kind of fuzzy search. Texts are tokenized (split into terms) and the fuzzy comparison is done term by term.
- For example, when searching with 'SAP', a record like 'SAP Deutschland AG & Co. KG' gets a high score, because the term 'SAP' exists in both texts. A record like 'SAPPHIRE NOW Orlando' gets a lower score, because 'SAP' is only a part of the longer term 'SAPPHIRE' (3 of 8 characters match).

Figure 294: Text Type Fuzzy Search

Date Type Searches

Fuzzy search on date values checks for date-specific errors. For example:

- Date-specific typos. For example, days or dates that have month and day exchanged such as American versus British date format.
- Dates lying within a user-defined maximum distance

Fuzzy search for dates works with valid dates only. A search with an invalid date does not return any results. The following example does not return any results, as **31** is not a valid month: `...WHERE CONTAINS(mydate, '2012-31-01', FUZZY(0.8))...`

The maximum allowed distance between dates can be defined using the search option `maxDateDistance`, which defines a number of days.

Fuzzy Search SQL Syntax

A fuzzy search can be performed by using a `SELECT` statement, by explicitly specifying the option `FUZZY` in the `CONTAINS` function. The fuzziness threshold can be passed as an optional parameter; otherwise, a default value **0.8** is used.

- You can call the fuzzy search by using the CONTAINS() function with the FUZZY() option in the WHERE-clause of a SELECT statement.

```
SELECT
        SCORE() AS score, *
FROM
        documents
WHERE
        CONTAINS(doc_content, 'Driethanolamyn', FUZZY(0.6))
ORDER BY
        score DESC;
```

- Optionally, the **fuzziness threshold** can manually be set when making the FUZZY() call.
- If set to for example 0.6, no matches lower than 0.6 will be returned. **Default is 0.8**.

Figure 295: Fuzzy Search SQL Syntax

Additional Features

You can also search for a string across multiple columns at once in a Freestyle search.

- If you want HANA to search for occurrences of your search word or phrase in multiple columns, you can perform a "Freestyle Search". This type of search will go through any columns in the table that has "Freestyle Search" enabled.

```
SELECT
        SCORE() AS score, *
FROM
        documents
WHERE
        CONTAINS(*, 'Driethanolamyn', FUZZY)
ORDER BY
        score DESC;
```

Replacing the column name with a star

Figure 296: Freestyle Search SQL Syntax

Further Full Text Search Syntax – Highlighting and Snippets

Highlighting – Mark the search terms in the text

```
SELECT HIGHLIGHTED(DEFECT), CAMP_NO FROM
"_SYS_BIC"."nhtsa/J_RECALLS_S" WHERE CONTAINS(DEFECT, 'throttle');
```

HIGHLIGHTED(DEFECT)	CAMP_NO
inner and outer carburetor throttle return springs may have been misinstalled possibly resulting in spring failure this can prevent the throttle from retu...	79V074000
under a combination of circumstances in cold weather icing may prevent return of the throttle butterfly to the idle position	79V020000
due to an improperly seated bushing in the carburetor secondary barrel body a portion of the spiral return spring of the throttle plate opener may restrict the fre...	78V262000
on the involved vehicles the vacuum line connecting the carburetor and the thermo valve may have been improperly installed this condition may prevent the <b>throttl...	77V214000
the possibility exists that the aluminum engine rear support brackets may fail due to metal fatigue this type of failure would allow the engine and transmission assembly t...	78V170000

Snippets – Returns a short snippet of the text with the search term highlighted

```
SELECT SNIPPETS(DEFECT), CAMP_NO FROM
"_SYS_BIC"."nhtsa/J_RECALLS_S" WHERE CONTAINS(DEFECT, 'throttle');
```

SNIPPETS(DEFECT)	CAMP_NO
inner and outer carburetor throttle return springs may have been misinstalled possibly ...	79V074000
... icing may prevent return of the throttle butterfly to the idle position	79V020000
... the spiral return spring of the throttle plate opener may restrict the ...	78V262000
... installed this condition may prevent the throttle linkage plate from returning to ...	77V214000
... dislocate downward this will result in throttle control loss with the engine ...	78V170000

Figure 297: Further Full Text Search Syntax – Highlighting and Snippets

Note:
In fuzzy search, the marked term is not necessarily identical to the search term.

Further Full Text Search Syntax – Why Found and Faceted Search

Why found – Information about where a search term was found

```
SELECT WHY_FOUND(), CAMP_NO FROM "_SYS_BIC"."nhtsa/J_RECALLS_S"
       WHERE CONTAINS(*, 'throttle engine')
```

WHY_FOUND	CAMP_NO
<TERM>engine</TERM><FOUND>DEFECT</FOUND><TERM>throttle</TERM><FOUND>CONSEQUENCE</FOUND><FOUND>DEFECT</FOUND>	05V172000
<TERM>engine</TERM><FOUND>DEFECT</FOUND><TERM>throttle</TERM><FOUND>DEFECT</FOUND>	10V015000
<TERM>engine</TERM><FOUND>CONSEQUENCE</FOUND><FOUND>DEFECT</FOUND><TERM>throttle</TERM><FOUND>DEFECT</FOUND>	09V471000
<TERM>engine</TERM><FOUND>CONSEQUENCE</FOUND><TERM>throttle</TERM><FOUND>DEFECT</FOUND>	10V327000
<TERM>engine</TERM><FOUND>CONSEQUENCE</FOUND><TERM>throttle</TERM><FOUND>CONSEQUENCE</FOUND><FOUND>DEFECT</FOUND>	09V360000
<TERM>engine</TERM><FOUND>DEFECT</FOUND><TERM>throttle</TERM><FOUND>DEFECT</FOUND>	07E053000
<TERM>engine</TERM><FOUND>CONSEQUENCE</FOUND><FOUND>DEFECT</FOUND><TERM>throttle</TERM><FOUND>CONSEQUENCE</FOUND><FOUND>DEFECT</FOUND>	03E038000

Faceted Search – Aggregated count for a column

```
SELECT MAKE_FACET, COUNT(*) AS C FROM
_SYS_BIC"."nhtsa/J_RECALLS" WHERE CONTAINS(DEFECT, 'throttle
engine') GROUP BY MAKE_FACET ORDER BY C DESC;
```

	MAKE_FACET	C
1	NISSAN	424
2	CHEVROLET	424
3	FORD	300
4	MACK	276
5	AIR BYPASS VALVE	216

Figure 298: Further Full Text Search Syntax – Why Found and Faceted Search

Unit 5
Exercise 23

Use Full Text Search

Exercise Objectives

After completing this exercise, you will be able to:

- Write SQL select statements making use of Full Text Search functions
- Set Fuzzy Search parameters inside SQL select statements

Business Example

You have a database table with customer data. The data quality is not accurate. You want to search for certain fields in your customer base even though you know it contains misspellings.

Task 1: Create the table that will be used to perform Full Text Search statements.

1. Make a copy of the following SQL files to your local N: drive (root):

   - 006_FUZZY_SEARCH.sql
   - 007_FUZZY_SEARCH.sql

   The SQL files are located in a folder that you can access using a shortcut in the Start menu.

   Choose *Start* in the Windows taskbar and search for: `student`

   - Folder shortcut name: *student — hanastudent — hanareadonly*
   - User name: `hanastudent`
   - Password: `hanareadonly`

2. Open the SQL file *HA300 → 006_FULL_TEXT_SEARCH.sql* in notepad and copy the contents to your clipboard.

3. Open a SQL Console by right-clicking on the HANA connection in the Systems view, choose *Open SQL Console*.

4. Paste the contents of your clipboard in the SQL Console.

5. Replace all occurrences of `##` in `student##` with your student number and execute the script after connecting the SQL editor to the SAP HANA database. Ensure that the newly created table *searchtbl_student##* is not empty.

   > Hint:
   > After creating the table, press F5 to refresh the *Tables* node in the *Catalog* to see it.

> Caution:
> After performing a search and replace in the SQL Console of the SAP HANA Studio, the last replaced string is highlighted. So if you execute the script as is, in most cases, you get a SQL error (except if the replaced string itself is a valid SQL statement).
>
> To avoid this, before executing the SQL script, remove the highlight by clicking anywhere in the SQL Console pane.

Task 2: Perform Fuzzy Searches

> Note:
> Before executing each of the SQL statements, make sure you understand the purpose of the statement.

1. Open the SQL file *007_FUZZY_SEARCH.sql* on your N:\ drive.
2. Paste the contents of the *007_FUZZY_SEARCH.sql* in the SQL console, replacing the existing content.
3. Replace all occurrences of `##` in `student##` with your student number.
4. Search for customers in counties that contains the word `West`.

   Using your mouse cursor, highlight the code from the beginning of the file up to the end of the *Step 1* statement and Execute this part of code.

   How many customers are returned? Note the answer in the following field:

   ____

   ____

   ____

5. Run a similar query, but this time, set the Fuzzy parameter to *0.3*.

   Execute the SQL statement *Step 2*.

   Why do you think customer *Cody Burnett* is included in the result set? Note the answer in the following field:

   ____

   ____

   ____

> Note:
> You can notice that the fuzziness threshold is not simply the SCORE. For example, a record with a *SCORE* of *0.258* is returned.

6. Perform a Fuzzy Search for customers working in a department that includes the word `Department`.

You want to use the *HIGHLIGHTED* function to see why non-exact matches are returned.

Execute the SQL Statement *Step 3*.

Review the first column to see why rows are returned. Highlighting occurs between markers ** and **.

7. Perform a Fuzzy Search on all columns, searching for the word **`Customer`**.

   Execute the SQL Statement *Step 4*.

   Which column do you think contains an entry generating a match for customer *Odysseus Larsen*?

   ____

   ____

   ____

8. Perform a similar search as per the previous step, but this time count the number of returned rows per customer industry (column: *CUST_INDUSTRY*).

   Execute the SQL Statement *Step 5*.

   How many customers matching the Fuzzy Search are found in the following industries? Note down your answers below:

Table 15: Fuzzy Search Matches

Industry	Number of Fuzzy Search Matches
Banking	
Retail	
Software	

Unit 5
Solution 23

Use Full Text Search

Exercise Objectives

After completing this exercise, you will be able to:

- Write SQL select statements making use of Full Text Search functions
- Set Fuzzy Search parameters inside SQL select statements

Business Example

You have a database table with customer data. The data quality is not accurate. You want to search for certain fields in your customer base even though you know it contains misspellings.

Task 1: Create the table that will be used to perform Full Text Search statements.

1. Make a copy of the following SQL files to your local N: drive (root):

   - 006_FUZZY_SEARCH.sql
   - 007_FUZZY_SEARCH.sql

   The SQL files are located in a folder that you can access using a shortcut in the Start menu.

   Choose *Start* in the Windows taskbar and search for: **`student`**

   - Folder shortcut name: *student — hanastudent — hanareadonly*
   - User name: **`hanastudent`**
   - Password: **`hanareadonly`**

   a) In the Windows taskbar, choose *Start* and search for: **`student`**.

   b) Locate the files.

   c) Copy and paste the files from the folder to your N: drive.

2. Open the SQL file *HA300 → 006_FULL_TEXT_SEARCH.sql* in notepad and copy the contents to your clipboard.

   a) Double-click the file *HA300 → 006_FULL_TEXT_SEARCH.sql*.

   b) Copy the contents to your clipboard.

3. Open a SQL Console by right-clicking on the HANA connection in the Systems view, choose *Open SQL Console*.

   a) In the modeler perspective of HANA Studio, right-click on the HANA connection in the *Systems* view and choose *Open SQL Console*.

4. Paste the contents of your clipboard in the SQL Console.

   a) To paste, use CTRL+V.

5. Replace all occurrences of **##** in **student##** with your student number and execute the script after connecting the SQL editor to the SAP HANA database. Ensure that the newly created table *searchtbl_student##* is not empty.

> Hint:
> After creating the table, press F5 to refresh the *Tables* node in the *Catalog* to see it.

> Caution:
> After performing a search and replace in the SQL Console of the SAP HANA Studio, the last replaced string is highlighted. So if you execute the script as is, in most cases, you get a SQL error (except if the replaced string itself is a valid SQL statement).
>
> To avoid this, before executing the SQL script, remove the highlight by clicking anywhere in the SQL Console pane.

a) Replace all occurrences of **##** in **student##** with your student number.

b) To execute the script, press F8.

c) In the *Systems* view, select the *Tables* node in your schema **STUDENT##** and press F5 to refresh the list.

d) Right-click the table *searchtbl_student##* and choose *Open Data Preview*.

Task 2: Perform Fuzzy Searches

> Note:
> Before executing each of the SQL statements, make sure you understand the purpose of the statement.

1. Open the SQL file *007_FUZZY_SEARCH.sql* on your N:\ drive.

   a) Using notepad, open the SQL file *007_FUZZY_SEARCH.sql* on your N:\ drive.

2. Paste the contents of the *007_FUZZY_SEARCH.sql* in the SQL console, replacing the existing content.

   a) Select everything in the SQL console and press CTRL+V.

3. Replace all occurrences of **##** in **student##** with your student number.

   a) Perform a search and replace of **##** with your student number.

4. Search for customers in counties that contains the word **West**.

   Using your mouse cursor, highlight the code from the beginning of the file up to the end of the *Step 1* statement and Execute this part of code.

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

How many customers are returned? Note the answer in the following field:

Customers Returned: ______

a) Highlight the portion of the SQL script as specified and press F8.

b) Answer: 4 customers are returned.

5. Run a similar query, but this time, set the Fuzzy parameter to *0.3*.

   Execute the SQL statement *Step 2*.

   Why do you think customer *Cody Burnett* is included in the result set? Note the answer in the following field:

   [Correct answer] ______

> Note:
> You can notice that the fuzziness threshold is not simply the SCORE. For example, a record with a *SCORE* of *0.258* is returned.

a) Highlight the portion of the SQL script as specified and press F8.

b) Answer: There are a number of matching characters when comparing the word **`Gwent`** to **`West`**.

6. Perform a Fuzzy Search for customers working in a department that includes the word **`Department`**.

   You want to use the *HIGHLIGHTED* function to see why non-exact matches are returned.

   Execute the SQL Statement *Step 3*.

   Review the first column to see why rows are returned. Highlighting occurs between markers ** and **.

a) Highlight the portion of the SQL script as specified and press F8.

b) Notice that the fuzziness threshold considers **`Development`** as a valid match for **`Department`**.

7. Perform a Fuzzy Search on all columns, searching for the word **`Customer`**.

   Execute the SQL Statement *Step 4*.

   Which column do you think contains an entry generating a match for customer *Odysseus Larsen*?

   Column Name: ______

a) Highlight the portion of the SQL script as specified and press F8.

b) Answer: The *CUST_COUNTY* column matches a number of characters when comparing the word **`Somerset`** to **`Customer`**.

8. Perform a similar search as per the previous step, but this time count the number of returned rows per customer industry (column: *CUST_INDUSTRY*).

   Execute the SQL Statement *Step 5*.

   How many customers matching the Fuzzy Search are found in the following industries? Note down your answers below:

   Table 15: Fuzzy Search Matches

Industry	Number of Fuzzy Search Matches
Banking	
Retail	
Software	

   a) Highlight the portion of the SQL script as specified and press F8.

   b) Answer:

   Table 16: Fuzzy Search Matches

Industry	Number of Fuzzy Search Matches
Banking	10
Retail	3
Software	1

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

LESSON SUMMARY

You should now be able to:

- Use Full Text Search
- Use Fuzzy Search
- Explain Fuzzy Search relevance scoring
- Use Freestyle Search

Unit 5
Lesson 4

Working with Geospatial Data

LESSON OVERVIEW

This lesson gives you an introduction about how to store, process, and publish geospatial data in SAP HANA.

Business Example

You want to calculate the distances between your customers and your stores.

Typically, locations or addresses in a database are stored in text fields, making it difficult to pinpoint the exact geographical locations.

To be able to calculate distances accurately, you have decided to store the geographical locations of your customers and stores, using the SAP HANA geospatial model and its related functions.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Work with Geospatial Data
- Query geographical data using SQL

Geospatial Data in SAP HANA

SAP HANA includes features making it possible to store locations in geographical datatypes.



These features are termed **spatial** or **geospatial**.

Spatial in this context refers to data pertaining to a geographical location.

Using the spatial features you are able to store, process, manipulate, share, and retrieve spatial data directly in the database

Figure 299: SAP HANA Geospatial Introduction

By utilizing geospatial features, it is possible to create database queries to answer questions such as:

- Which of my customers are close to a certain store?

- Which is the closest available shipment hub?
- Which of my regions do not have enough stores to support its customers?

Geospatial Features as Part of the SAP HANA Platform

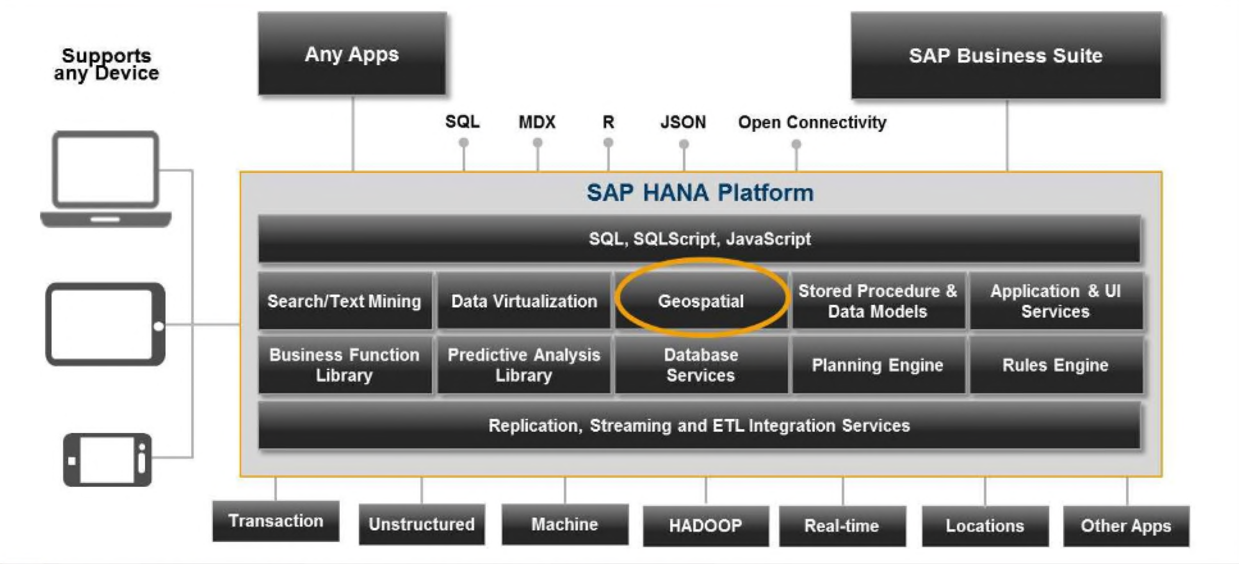


Figure 300: Geospatial Features as Part of the SAP HANA Platform

SAP HANA includes a multi-layered Spatial Engine and supports spatial columns that can be referenced using spatial access methods.

Spatial data is data that describes the position, shape, and orientation of objects in a defined space. Spatial data is represented as 2D or 3D geometries in the form of points, line strings, and polygons.

Spatial Processing Architecture

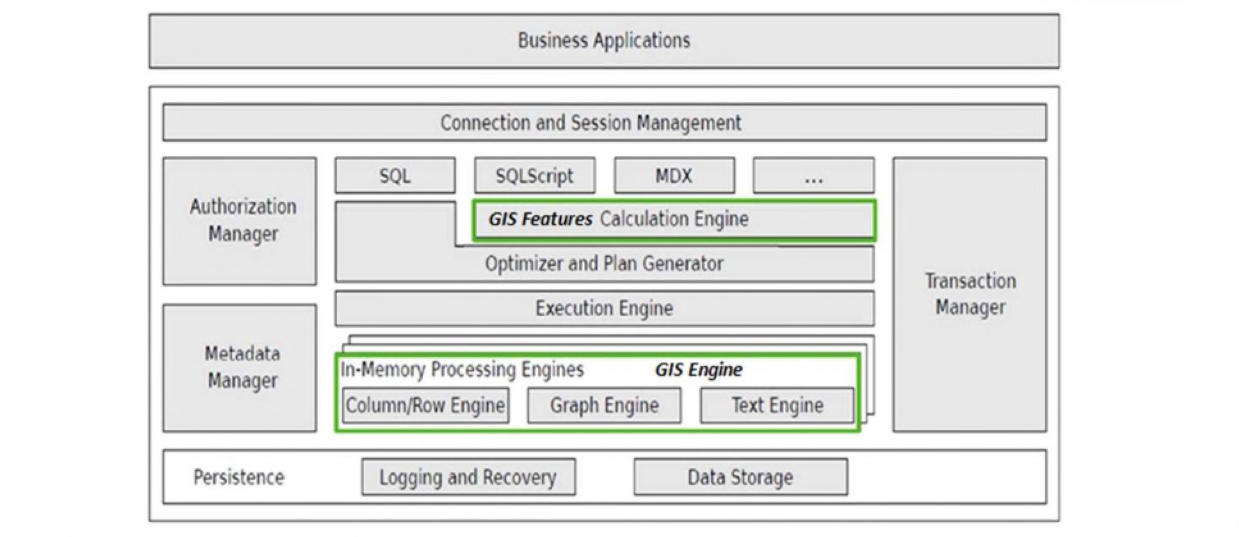


Figure 301: Spatial Processing Architecture

The Spatial Engine adds new functionality to the Calculation Engine. It does not constitute a new type of engine, like, for example, the Join Engine.

The spatial features that form part of the SAP HANA architecture support the following:

- 2D and 3D Vector Types
- Points, line-strings, polygons, compound polygons
- SRID (Spatial Reference IDs)
- Application development using the XS engine with geo-content and mapping services

> Note:
> SAP HANA offers the possibility to express spatial coordinates based on a specific Plan-Earth or Round-Earth reference system. For example, the WGS84 (World Geodetic System), which is the one used by the GPS (Global Positioning System).

Spatial Data as Part of the Column Store

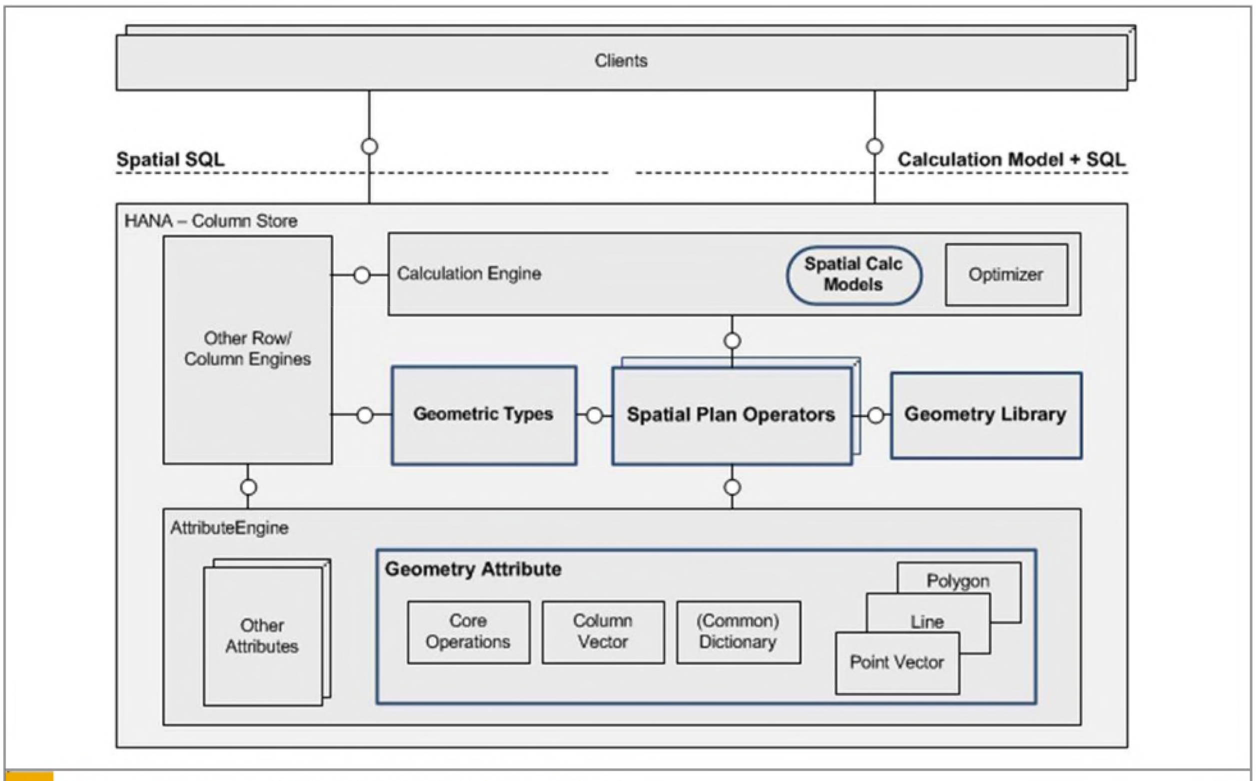


Figure 302: Spatial Data as Part of the Column Store

Geographical queries are processed by the native Spatial Engine, which is part of the Calculation Engine, which in turn is embedded in the Indexserver.

Spatial data is stored in the column store in geometric attributes, using optimized data types such as ST_GEOMTERY and ST_POINT.

Geographical data can be called using spatial plan operators, which are embedded into the SAP HANA SQL. These plan operators can be part of calculation and information models or can be called using SQL queries directly on tables.

SQLScript also supports spatial data types and functions so that functions can be created and procedures called to access and manipulate spatial data.

Geographical Data Types

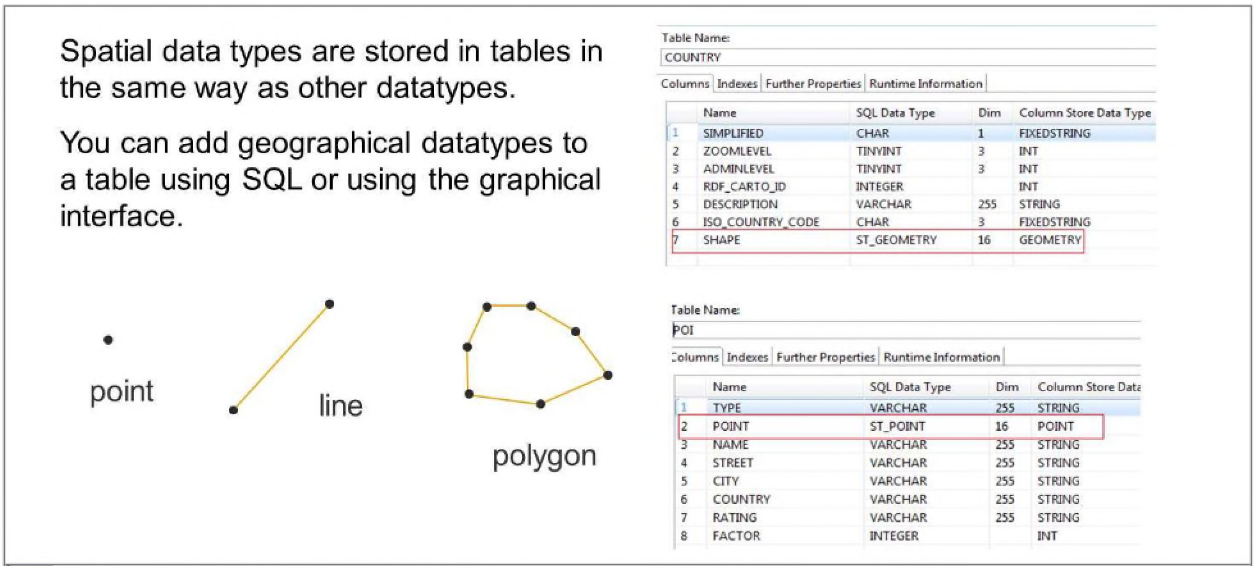


Figure 303: Geographical Data Types

SAP HANA Spatial Data Types

The following data types are used for spatial data:

Data Type	Use Case
ST_GEOMETRY	The geometry type is the supertype for all supported spatial data types.
ST_POINT	A point defines a single location in space. A point geometry does not have length or area. A point always has an X and Y coordinate.
ST_POLYGON	A polygon defines a region of space. In GIS data, polygons are typically used to represent territories (countries, towns, states).
ST_LINESTRING	A linestring is geometry with a length, but without any area. In GIS data, linestrings are typically used to represent rivers, roads, or delivery routes.
ST_MULTILINE	A multilinestring is a collection of linestrings. In GIS data, multilinestrings are often used to represent geographic features like a highway network.
ST_MULTIPOINT	A multipoint is a collection of individual points.
ST_MULTIPOLYGON	A multipolygon is a collection of zero or more polygons. In GIS data, multipolygons are often used to represent territories made up of multiple regions (for example a state with islands).
ST_CIRCULARSTRING (as of SAP HANA SPS09)	A circularstring uses circular line segments between control points. Three points (start, intermediary, and end) determine a circularstring. You can then add another circular line segment with an additional intermediary and end point.

> Note:
> A circle is a particular case of circularstring where the start and end point are identical.

ST_POINT Data Type

The shape of the geographical data you want to store determines which data type you will use.

Geospatial data is typically entered using the WKT (Well Known Text) format. This format is maintained by the Open Geospatial Consortium (OGC) as part of the Simple Features defined for the OpenGIS Implementation Specification for Geographic Information. (see http://www.opengeospatial.org/standards/sfa)

> Note:
> Between 1994 and 2004, OGC was also known as Open GIS Consortium.

Here is an example of how a point might be represented in WKT:

```
'POINT(1 1)'
```

There are other ways of entering and retrieving geographical data as follows:

- WKB (Well Known Binary)
- GeoJSON
- ESRI shapefiles (a popular geospatial vector data format)

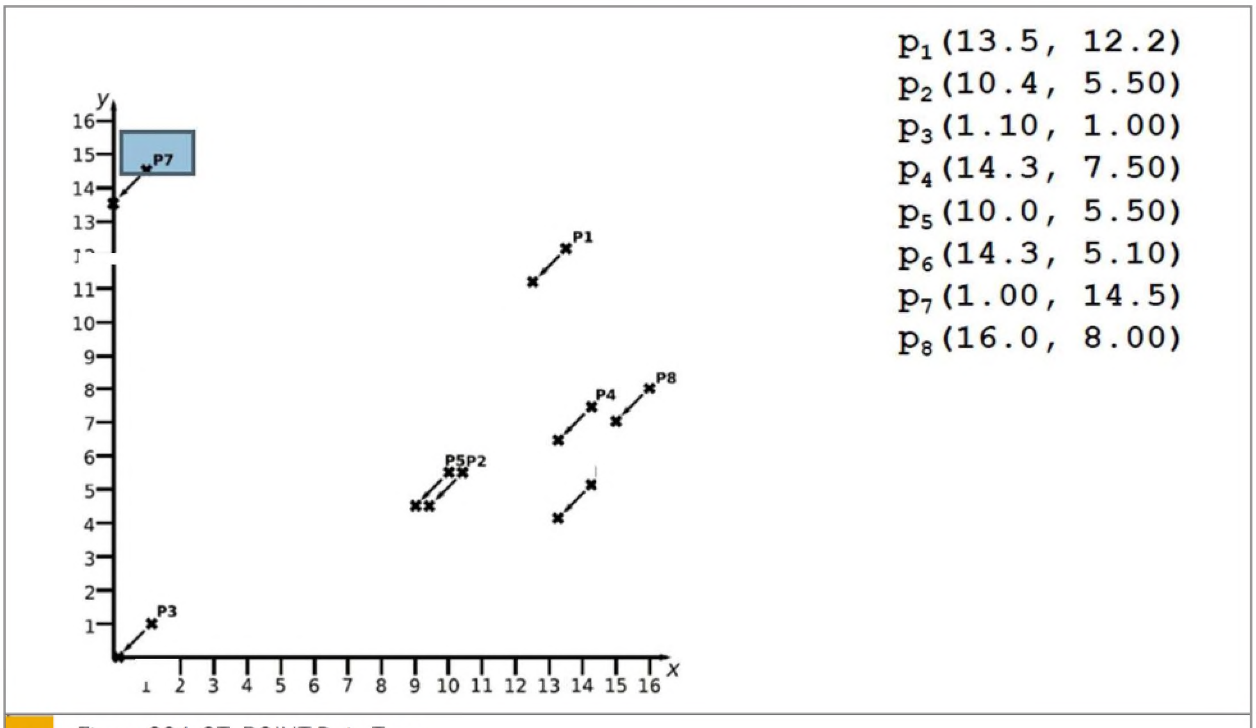


Figure 304: ST_POINT Data Type

The most simple example of a spatial data type is **ST_POINT**, which stores geographical locations in using X and Y coordinates.

> Note:
> From SAP HANA 1.0 SPS08 onwards, SAP HANA supports 3D coordinates. Therefore, any point can also have a Z coordinate.

Spatial Data Functions

SAP HANA offers a number of functions to populate spatial columns from different sources (Well Known Text –WKT– and Well Known Binary –WKB), and to retrieve the spatial data as one of these different formats. Examples include the following:

- To create geometries:
  - ST_GeomFromText
  - ST_GeomFromWKT
  - ST_GeomFromWKB
- To retrieve spatial data as one particular format:
  - ST_AsWKT
  - ST_AsWKT
  - ST_AsWKB

Other functions provide information about geometries to answer questions such as the following:

- In which SRID (Spatial Reference ID) are the coordinates are expressed?
- Is a geometry valid?
- Is a geometry 3D?

Creating, Retrieving, and Comparing Spatial Data in SAP HANA Tables

To create a table including a column with a spatial data type and then run queries on such columns, you use the syntax in the figure, Spatial Column Creation and Spatial Data Access. It is also possible to add a spatial column to an existing table.

1. Table creation

```
CREATE COLUMN TABLE locations
(
   id          INTEGER,
   description CHAR(100),
   location    ST_POINT
);
```

2. Spatial SQL access

```
SELECT
  description
FROM
  locations
WHERE
  locations.ST_EQUALS('POINT(3.0 6.0)') = 1
```

Figure 305: Spatial Column Creation and Spatial Data Access

The second step of the example uses a Spatial function, **ST_EQUALS**.

The example uses a spatial function in a **WHERE** clause to restrict a query to only return rows where the location is equal to the (X,Y) coordinate (3.0,6.0).

Functions for Spatial Data Comparison

To use geographical locations in a meaningful way, very often you have to compare spatial geometries.

SAP HANA provides spatial functions that can query spatial columns using SQL.

Examples of some spatial functions, their syntax, and when they can be used are as follows:

Table 17: Spatial Data Comparison Function

Spatial Function and Syntax	Usage
`[geo1].ST_EQUALS([geo2])`	Tests if a ST_GEOMETRY value is spatially equal to another ST_GEOMETRY value.
`[geo1].ST_CONTAINS([geo2])` and `[geo1].ST_COVERS([geo2])`	Tests if a geometry value spatially contains another geometry value.*
`[geo1].ST_WITHIN([geo2])` and `[geo1].ST_COVEREDBY([geo2])`	Tests if a geometry value is spatially contained within another geometry value.*
`[geo1].ST_WITHINDISTANCE([geo2], [distance])`	Test if two geometries are within a specified distance of each other.
`[geo1].ST_DISTANCE([geo2])`	Returns the smallest distance between specified geometry values.

* The **ST_COVERS** and **ST_CONTAINS** methods are almost similar. The differences are as follows:

- **ST_CONTAINS** evaluates to 1 (TRUE) only if one or more interior points of [geo2] lie in the interior of [geo1].

  Similarly, **ST_COVEREDBY** evaluates to 1 (TRUE) only if one or more interior points of [geo1] lie in the interior of [geo2].
- **ST_COVERS** and **ST_COVEREDBY** methods can be used with geometries in round-Earth spatial references, while **ST_CONTAINS** and **ST_WITHIN** only apply to Flat-Earth reference systems.

Spatial Clustering

SAP HANA supports algorithms to group spatial points into set of clusters.

- **Spatial Clustering** groups a set of points, that meets certain criteria, into clusters
  - A cluster is a partition of the original set of points
  - Spatial clustering essentially allows geographical information to be used to group data
- Example use cases:

Retail Store Location
•Retailer wants to establish two chains of stores
•One luxury brand, one discount
•Has location & income data of 120M US households

Goal:
•Locate areas of high income and low income households to strategically locate the two different chains

Vending Outlets
•Retailer has vending machines located across the US
•Has location and revenue generated of each machine

Goal:
•Analyze lucrative and non-lucrative locations
•Easily visualize these locations
•Do ad-hoc analysis – e.g. zoom in and get specific machine details

Figure 306: Spatial Clustering Overview

Spatial Clustering — Supported Algorithms

There are three different types of algorithms available to group spatial data, depending on the use case.

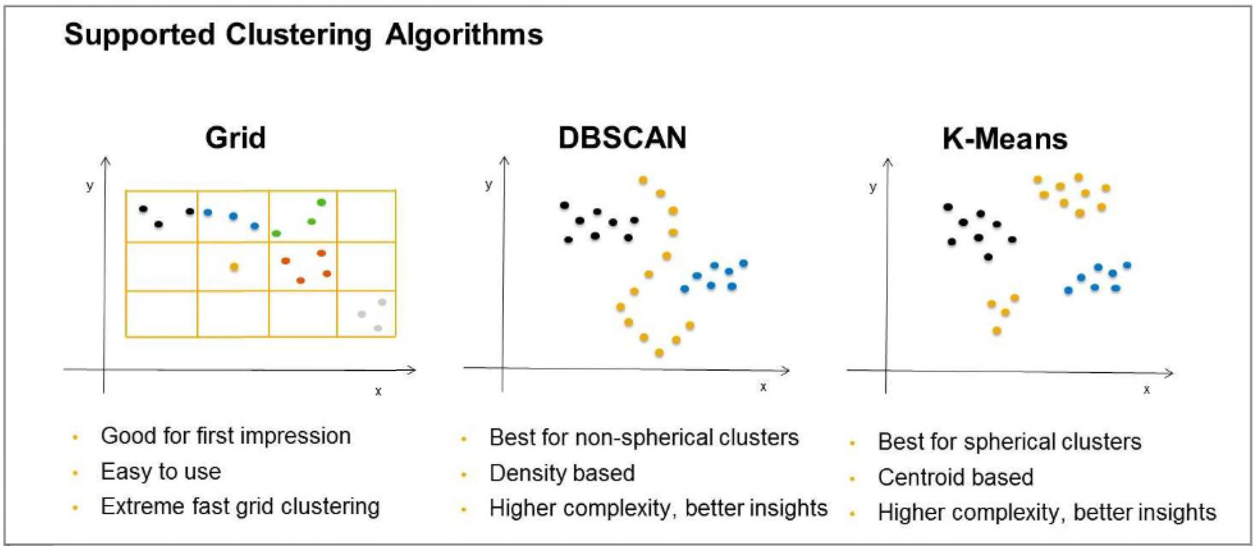


Figure 307: Spatial Clustering — Supported Algorithms

Spatial Clustering Example using K-means Algorithm

Once data is grouped into clusters, it is possible to graphically visualize the results using Extended Application Services (XS).

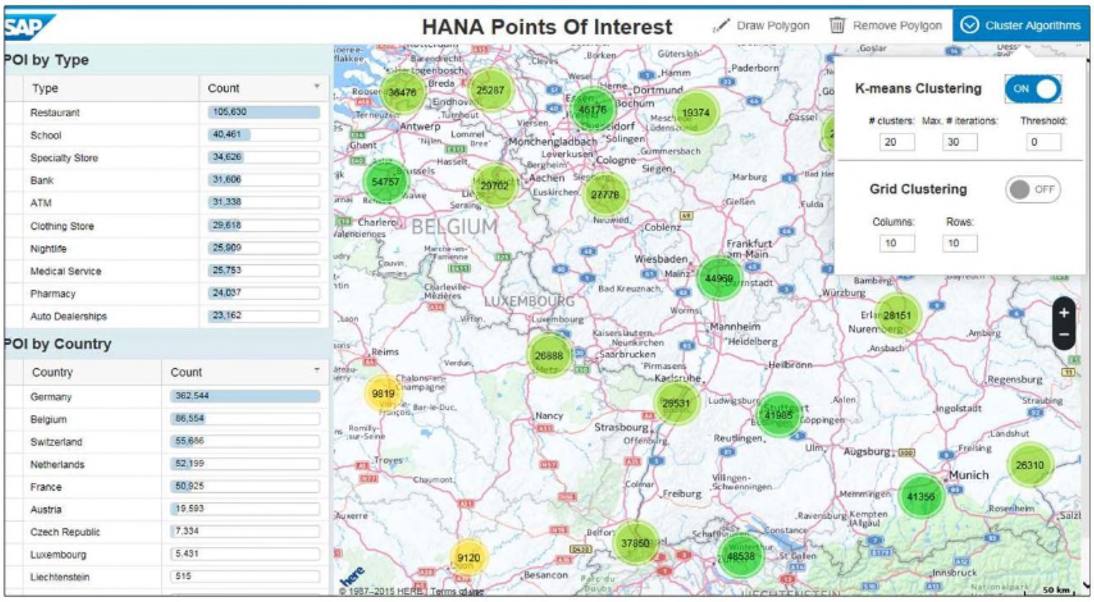


Figure 308: Spatial Clustering Example using K-means Algorithm

Spatial Clustering Example

Spatial clustering can be performed within SQL statements, such as in the following example:

```
SELECT
 ST_ClusterId() AS cluster_id,
 ST_ClusterCentroid() AS centroid,
 COUNT(*) AS number_of_households,
 AVG(INCOME) AS average_cluster_income
FROM HOUSEHOLDS
WHERE INCOME > 120000
GROUP CLUSTER BY LOCATION USING KMEANS CLUSTERS 100
HAVING COUNT(*) >= 300;
```

Here, household income data is grouped into clusters using a K-means algorithm.

Geocoding Features

Geocoding is the process of converting address information into geographic locations. This is crucial when you want to apply geospatial processing to your data if it does not initially contain spatial coordinates.

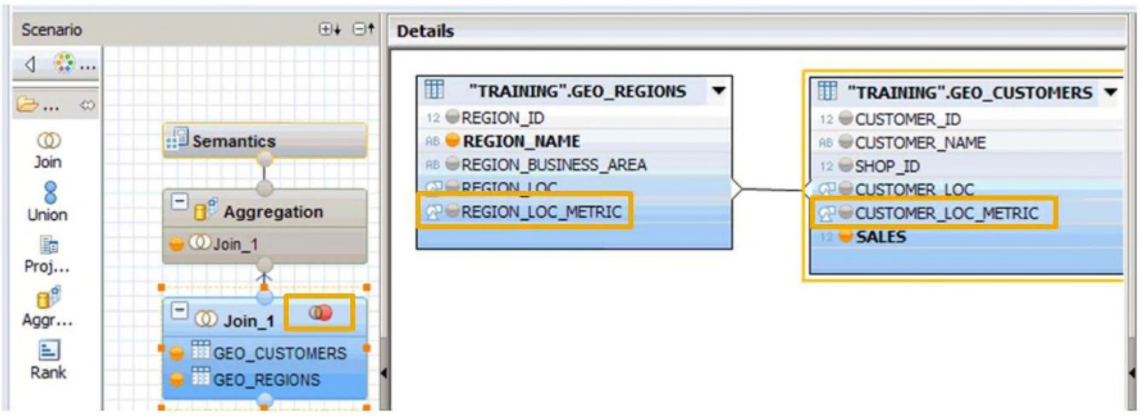
SAP HANA comes with a feature to create a dedicated index (of type GEOCODE) on a table that contains the text-based address. Each of the columns containing address information must be specified.

The geocoding itself will be provided by an external provider (for example, Nokia NAVTEQ or TOMTOM), and the outbound connectivity is provided by the Extended Application Services (XS).

This index is updated asynchronously, and it is a visible column that you can use directly in queries with spatial functions.

Using Spatial Joins in Calculation Views and Core Data Services (CDS)

Spatial joins can be utilized in graphical Calculation Views



It is also possible to call Spatial functions using SQL Script.

Additionally, Core Data Services (CDS) supports spatial data types and functions

Figure 309: Spatial Functions in Calculation Views and CDS

SAP HANA gives the ability to use spatial joins in Calculation Views. This allows you to define easily a semantic comparison of two geometries (for example, is a point contained within–or covered by–a polygon). The Core Data Services (CDS) now also support spatial data types and functions.

> **Note:**
> In versions prior to SAP HANA SPS09, this type of comparison could only be performed by SQL in an SQL Editor.

Spatial Joins in Calculation Views

- A spatial data join behaves as a spatial function that compares two geographies.
- In this example, the Calculation View will match customer and region only if the customer's location is included in the region (Within).

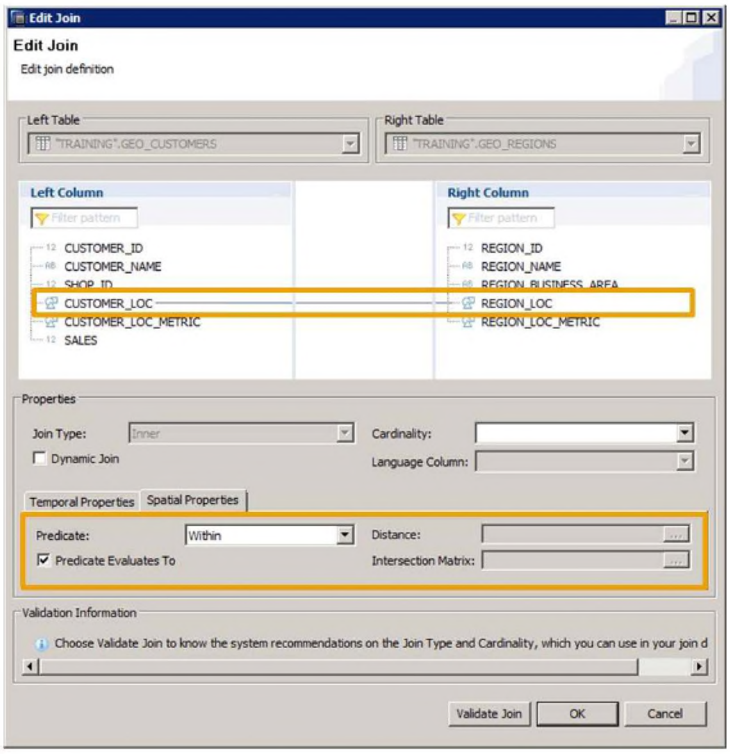


Figure 310: Spatial Joins in Calculation Views

Calculation Columns and Filters Using Spatial Expressions

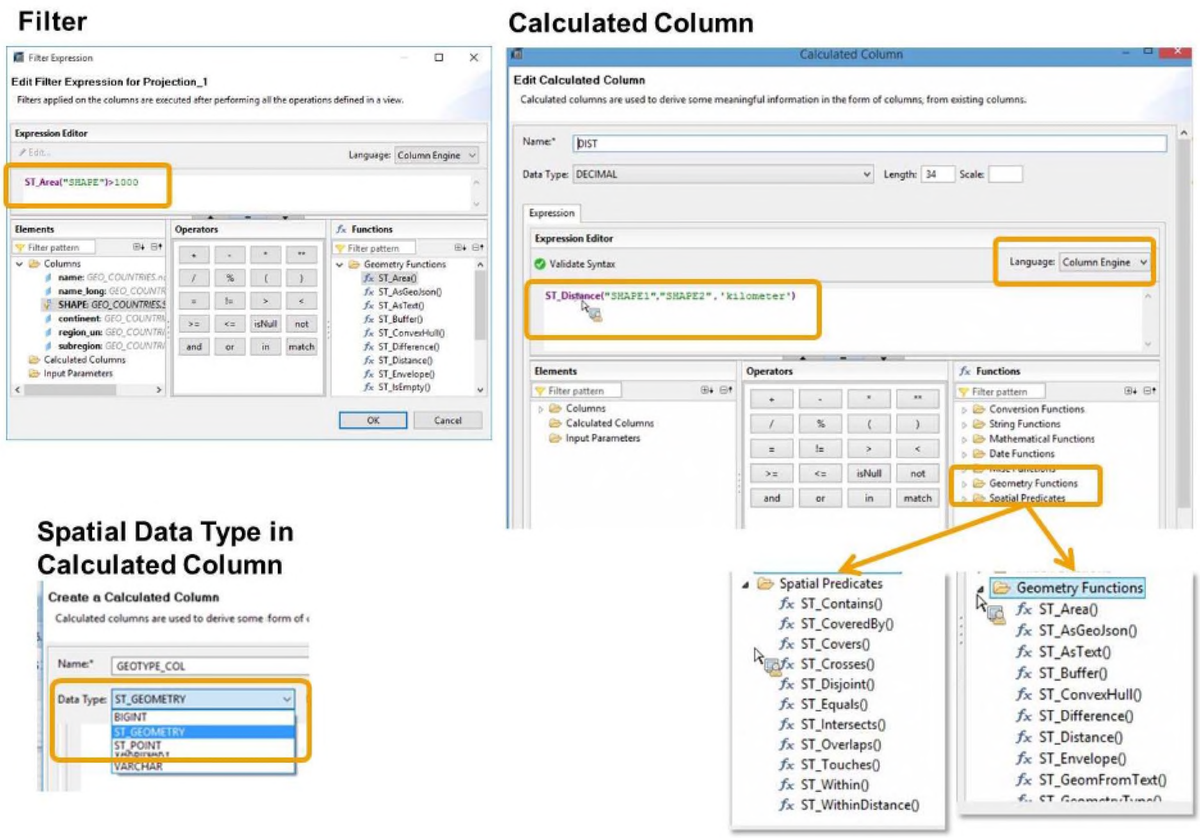


Figure 311: Spatial Functions in Calculated Columns and Filters

SAP HANA SPS12 brings the possibility to create calculated columns of spatial data types **ST_GEOMETRY** and **ST_POINT**.

In addition to spatial predicates that were already available, such as **ST_Contains()**, **ST_Intersects()**, calculated columns now support spatial functions, for example, **ST_Area()**, **ST_Intersection()**, as well as conversion functions, for example **ST_asText()** and **ST_GeomFromText()**.

With SPS12, it is also possible to use spatial expressions in filters. For example, you can filter geometries based on their area.

Other Use Cases for Spatial Functions



Figure 312: Spatial Data Use Cases

While geography is the first use case that comes to mind, the Spatial Engine is able to handle arbitrary geometries.

The reference system that is indicated may be anything. Therefore, it becomes possible to provide geometry information, for example, for the human body, to visualize data and numbers at the correct place, instead of presenting a plain list.

Other uses cases include industrial design, or anything else that can be represented with coordinates in a grid.

Unit 5
Exercise 24

Use Geospatial Methods and Predicates

Exercise Objectives

After completing this exercise, you will be able to:

- Query geospatial data types
- Use geospatial methods in SQL
- Use Spatial Joins in a Calculation View

Business Example

You are managing an iron tool chain with shops located across the globe. The geographical location of your shops, regions and customers have been made available in SAP HANA. You want to create queries to answer geographical question such as, do some of my customers not have a shop close to their location?

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.

Task 1: Familiarize Yourself with the Geographical Tables used for the Exercise

Table 18: TRAINING.GEO_CUSTOMERS

Column	Datatype	Description
CUSTOMER_ID	INTEGER	Customer ID
CUSTOMER_NAME	VARCHAR	Customer full name
SHOP_ID	INTEGER	Shop ID of customer
CUSTOMER_LOC	ST_POINT	Geo coordinates of customer
CUSTOMER_LOC_METRIC	ST_POINT	Geo coordinates of customer in metric
SALES	INTEGER	Customer sales

Table 19: TRAINING.GEO_SHOPS:

Column	Datatype	Description
SHOP_ID	INTEGER	Shop ID
SHOP_NAME	VARCHAR	Shop name
SHOP_COUNTRY	VARCHAR	Country of shop

Column	Datatype	Description
SHOP_LOC	ST_POINT	Geo coordinates of shop
SHOP_LOC_METRIC	ST_POINT	Geo coordinates of shop in metric
NUMBER_OF_STAFF	INTEGER	Number of staff in shop

Table 20: TRAINING.GEO_REGIONS

Column	Datatype	Description
REGION_ID	INTEGER	Region ID
REGION_NAME	VARCHAR	Region name
REGION_BUSINESS_AREA	VARCHAR	Business area
REGION_LOC	ST_POLYGON	Geo polygon coordinates covering region
REGION_LOC_METRIC	ST_POLYGON	Geo polygon coordinates covering region in metric

Note:
All the SQL scripts for this exercise can be found in the file `Geospatial_SQL_Scripts.txt`. This file is located in your *Student* folder, within the *HA300* subfolder.

1. Open an SQL Editor and type the following SQL:

```
SELECT
customer_name,
shop_name
FROM
training.geo_customers
INNER JOIN training.geo_shops ON
geo_customers.shop_id = geo_shops.shop_id
```

2. Execute the SQL statement.

3. How many rows are returned?

____

____

____

Task 2: Enhance the Query with Geographical Methods and Predicates

1. Add a third column to the `SELECT` clause of your query showing the metric distance between each customer and their respective shop.

> Hint:
> Use the method `ST_Distance` on the metric location columns using syntax:
> ```
> [COLUMN1].ST_Distance([COLUMN2])
> ```

> Note:
> Since you are calculating the metric distance, use the metric location columns.

2. Add an `ORDER BY` clause and order the query in descending order by distance.
3. Execute the query.
4. Which customer is returned in the first row of the result set and thereby has the longest distance to their respective shop?

Task 3: Query Total Sales for Customers with a Distance to their Respective Shop of More than 100 KM (100,000 meters)

1. Remove the `ORDER BY` clause
2. Remove all columns from the `SELECT` clause and replace with a `SUM` of sales.
3. Add a `WHERE` clause restricting the query to customers with a distance of more than 100,000 meters to their shop.

> Hint:
> You can again use the `ST_Distance` function, this time as a predicate in the `WHERE` clause, using the same syntax:
> ```
> [COLUMN1].ST_Distance([COLUMN2])
> ```

4. Execute the query.
5. What is the total sales?

Task 4: Query the Total Sales of Customers Located in Business Area NSAM (North and South America)

1. Remove the join to the *GEO_SHOPS* table.
2. Remove the `WHERE` clause restricting to customer of a distance to their shop of more than 100,000 meters.
3. Add the `GEO_REGIONS` table to the FROM clause without a join.

4. Restrict the query so as to only show customers in the NSAM business area.

> Hint:
> Use the `ST_Within` predicate in the `WHERE` clause, using the following syntax:
>
> ```
> [COLUMN1].ST_Within([COLUMN2])
> = 1
> ```
>
> This time, do not use the metric location columns as this is not supported by the `ST_Within` function.

> Note:
> Using the `ST_Within` predicate in the `WHERE` clause will not be enough, because you want to retrieve only customers from the NSAM area. So you also need to add a restriction on the `GEO_REGIONS` table to show the `NSAM` region only.

5. Execute the query.

> Hint:
> Your final SQL should be similar to:
>
> ```
> SELECT
> SUM(sales)
> FROM
> training.geo_customers,
> training.geo_regions
> WHERE
> geo_regions.region_business_area = 'NSAM'
> AND customer_loc.ST_Within(region_loc) = 1
> ```

6. What is the total sales?

Task 5: Create a Calculation View to Retrieve the Total Sales per Business Area Based on a Spatial Join

Build a Calculation View that retrieves the total sales per business area, based on a spatial join that matches each customer with the region it belongs to.

1. Create a new graphical calculation view with the following properties:

Field	Value
Name	**`CV_SALES_REGION_##`**
Description	**`Total sales per region`**
Subtype	*Standard*

Field	Value
Data Category	*CUBE*

2. Create a new *Join* node and add the following tables to the new node: **GEO_CUSTOMERS** and **GEO_REGIONS**.

   Define the new node as the data source for the upper *Aggregation* node.

   Add the following columns to the output:

Table	Column
GEO_REGIONS	REGION_BUSINESS_AREA (attribute) REGION_NAME (attribute)
GEO_CUSTOMERS	SALES (measure; should be aggregated in the *Aggregation* node)

> Hint:
> To avoid adding the attribute columns to the output of both the *Join* and *Aggregation* nodes, you can directly select the required attributes (in the specified order), then right-click and choose *Propagate to Semantics*.
>
> For the *SALES* column (measure), add it to the output of the *Join* node and then to the output of the *Aggregation* node (as an aggregated column).

   Check that the columns are correctly defined in the Semantics of the Calculation View.

3. Define a spatial join between the fields *GEO_CUSTOMERS.CUSTOMER_LOC* and *GEO_REGIONS.REGION_LOC*.

   Define the *GEO_CUSTOMERS* table as the left one, and use the spatial predicate *Within*.

> Note:
> In case you define the left and right tables the other way round (that is, table *GEO_REGIONS* as the left table), just change the spatial predicate to *Contains*.

4. Save, Validate, and Activate the view and preview the data.

   Check that the total sales for region *NSAM* are consistent with the amount you retrieved in the previous task.

Unit 5
Solution 24

Use Geospatial Methods and Predicates

Exercise Objectives

After completing this exercise, you will be able to:

- Query geospatial data types
- Use geospatial methods in SQL
- Use Spatial Joins in a Calculation View

Business Example

You are managing an iron tool chain with shops located across the globe. The geographical location of your shops, regions and customers have been made available in SAP HANA. You want to create queries to answer geographical question such as, do some of my customers not have a shop close to their location?

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.

Task 1: Familiarize Yourself with the Geographical Tables used for the Exercise

Table 18: TRAINING.GEO_CUSTOMERS

Column	Datatype	Description
CUSTOMER_ID	INTEGER	Customer ID
CUSTOMER_NAME	VARCHAR	Customer full name
SHOP_ID	INTEGER	Shop ID of customer
CUSTOMER_LOC	ST_POINT	Geo coordinates of customer
CUSTOMER_LOC_METRIC	ST_POINT	Geo coordinates of customer in metric
SALES	INTEGER	Customer sales

Table 19: TRAINING.GEO_SHOPS:

Column	Datatype	Description
SHOP_ID	INTEGER	Shop ID
SHOP_NAME	VARCHAR	Shop name
SHOP_COUNTRY	VARCHAR	Country of shop

Column	Datatype	Description
SHOP_LOC	ST_POINT	Geo coordinates of shop
SHOP_LOC_METRIC	ST_POINT	Geo coordinates of shop in metric
NUMBER_OF_STAFF	INTEGER	Number of staff in shop

Table 20: TRAINING.GEO_REGIONS

Column	Datatype	Description
REGION_ID	INTEGER	Region ID
REGION_NAME	VARCHAR	Region name
REGION_BUSINESS_AREA	VARCHAR	Business area
REGION_LOC	ST_POLYGON	Geo polygon coordinates covering region
REGION_LOC_METRIC	ST_POLYGON	Geo polygon coordinates covering region in metric

> Note:
> All the SQL scripts for this exercise can be found in the file `Geospatial_SQL_Scripts.txt`. This file is located in your *Student* folder, within the *HA300* subfolder.

1. Open an SQL Editor and type the following SQL:

```
SELECT
customer_name,
shop_name
FROM
training.geo_customers
INNER JOIN training.geo_shops ON
geo_customers.shop_id = geo_shops.shop_id
```

   a) In the systems view on the left hand side of SAP HANA Studio, right-click the root node, and from the context menu, select *SQL Console*.

   b) Type the SQL as shown above.

   c) Alternatively, copy the SQL statement from the file mentioned in the note, and paste it into the SQL console.

2. Execute the SQL statement.

   a) Choose *Execute*, or use keyboard shortcut F8 to execute the SQL statement.

3. How many rows are returned?

   [Correct answer]

a) Answer: 12 rows are returned.

Task 2: Enhance the Query with Geographical Methods and Predicates

1. Add a third column to the `SELECT` clause of your query showing the metric distance between each customer and their respective shop.

> Hint:
> Use the method `ST_Distance` on the metric location columns using syntax:
> `[COLUMN1].ST_Distance([COLUMN2])`

> Note:
> Since you are calculating the metric distance, use the metric location columns.

a) Enhance the SQL by editing as follows:

```
SELECT
customer_name,
shop_name,
customer_loc_metric.ST_Distance(shop_loc_metric)
FROM
training.geo_customers
INNER JOIN training.geo_shops ON
geo_customers.shop_id = geo_shops.shop_id
```

2. Add an `ORDER BY` clause and order the query in descending order by distance.

a) Add the following `ORDER BY` clause to the end of your query:

```
ORDER
BY
customer_loc_metric.ST_Distance(shop_loc_metric) DESC
```

3. Execute the query.

a) Choose *Execute* or use keyboard shortcut F8 to execute the SQL.

4. Which customer is returned in the first row of the result set and thereby has the longest distance to their respective shop?

[Correct answer]

a) Answer: Madison Moore

Task 3: Query Total Sales for Customers with a Distance to their Respective Shop of More than 100 KM (100,000 meters)

1. Remove the `ORDER BY` clause

a) Remove the following from the SQL:

```
ORDER
BY
customer_loc_metric.ST_Distance(shop_loc_metric) DESC
```

2. Remove all columns from the `SELECT` clause and replace with a `SUM` of sales.

a) Remove the following from the SQL:

```
customer_name,

shop_name,
customer_loc_metric.ST_Distance(shop_loc_metric)
```

b) Write the following in code in place of the code that you just removed.

```
SUM(sales)
```

3. Add a `WHERE` clause restricting the query to customers with a distance of more than 100,000 meters to their shop.

> Hint:
> You can again use the `ST_Distance` function, this time as a predicate in the `WHERE` clause, using the same syntax:
>
> ```
> [COLUMN1].ST_Distance([COLUMN2])
> ```

a) Add the following `WHERE` clause to the end of your query:

```
WHERE

customer_loc_metric.ST_Distance(shop_loc_metric) > 100000
```

4. Execute the query.

a) Choose the execute button or use keyboard shortcut F8 to execute the SQL.

5. What is the total sales?

[Correct answer]

a) Answer: 1,851,437

Task 4: Query the Total Sales of Customers Located in Business Area NSAM (North and South America)

1. Remove the join to the *GEO_SHOPS* table.

a) Remove the following from the SQL:

```
INNER
JOIN training.geo_shops ON
geo_customers.shop_id = geo_shops.shop_id
```

2. Remove the `WHERE` clause restricting to customer of a distance to their shop of more than 100,000 meters.

a) Remove the following from the SQL:

```
customer_loc_metric.ST_Distance(shop_loc_metric)
> 100000
```

3. Add the `GEO_REGIONS` table to the FROM clause without a join.

a) Edit the `FROM` clause as shown in the following code:

```
training.geo_customers,
training.geo_regions
```

4. Restrict the query so as to only show customers in the NSAM business area.

Hint:
Use the `ST_Within` predicate in the `WHERE` clause, using the following syntax:

```
[COLUMN1].ST_Within([COLUMN2])
= 1
```

This time, do not use the metric location columns as this is not supported by the `ST_Within` function.

Note:
Using the `ST_Within` predicate in the `WHERE` clause will not be enough, because you want to retrieve only customers from the NSAM area. So you also need to add a restriction on the `GEO_REGIONS` table to show the `NSAM` region only.

a) Edit the `WHERE` clause to the following:

```
WHERE
geo_regions.region_business_area = 'NSAM'
AND customer_loc.ST_Within(region_loc) = 1
```

5. Execute the query.

Hint:
Your final SQL should be similar to:

```
SELECT
SUM(sales)
FROM
training.geo_customers,
training.geo_regions
WHERE
geo_regions.region_business_area = 'NSAM'
AND customer_loc.ST_Within(region_loc) = 1
```

a) Choose the execute button or use keyboard shortcut F8 to execute the SQL.

6. What is the total sales?

[Correct answer] ____________________

a) Answer: 1,398,440

Task 5: Create a Calculation View to Retrieve the Total Sales per Business Area Based on a Spatial Join

Build a Calculation View that retrieves the total sales per business area, based on a spatial join that matches each customer with the region it belongs to.

1. Create a new graphical calculation view with the following properties:

Field	Value
Name	**CV_SALES_REGION_##**

Field	Value
Description	**Total sales per region**
Subtype	*Standard*
Data Category	*CUBE*

a) Right-click your *STUDENT##* package and choose *New* → *Calculation View*.

b) Enter the details as shown in the table.

c) Choose *Finish*.

2. Create a new *Join* node and add the following tables to the new node: **GEO_CUSTOMERS** and **GEO_REGIONS**.

   Define the new node as the data source for the upper *Aggregation* node.

   Add the following columns to the output:

Table	Column
GEO_REGIONS	REGION_BUSINESS_AREA (attribute) REGION_NAME (attribute)
GEO_CUSTOMERS	SALES (measure; should be aggregated in the *Aggregation* node)

> Hint:
> To avoid adding the attribute columns to the output of both the *Join* and *Aggregation* nodes, you can directly select the required attributes (in the specified order), then right-click and choose *Propagate to Semantics*.
>
> For the *SALES* column (measure), add it to the output of the *Join* node and then to the output of the *Aggregation* node (as an aggregated column).

Check that the columns are correctly defined in the Semantics of the Calculation View.

a) Drag a *Join* node from the palette to the blank area below the Aggregation node.

   Optionally, choose *Auto Layout* in the bottom right corner of the *Scenario* pane.

b) Right-click the new *Join_1* node and choose *Add Objects*.

c) In the *Find* screen, enter **GEO_**. Select the two tables *GEO_CUSTOMERS* and *GEO_REGIONS* and choose *OK*.

d) To define the node *Join_1* as the data source for the *Aggregation* node, draw the corresponding connector by using drag-and-drop with your mouse.

e) In the *Join_1* node, select the two attributes from the *GEO_REGIONS* table, in the specified order (*REGION_BUSINESS_AREA* and *REGION_NAME*), then right-click and choose *Propagate to Semantics*.

   Then right-click the *SALES* column and choose *Add To Output*.

f) In the *Aggregation* node, right-click the *SALES* column and choose *Add as Aggregated Column*.

g) In the *Semantics* node, to flag the attributes, choose *Auto Assign* on the right.

Check that the semantics are correctly defined.

3. Define a spatial join between the fields *GEO_CUSTOMERS.CUSTOMER_LOC* and *GEO_REGIONS.REGION_LOC*.

Define the *GEO_CUSTOMERS* table as the left one, and use the spatial predicate *Within*.

> Note:
> In case you define the left and right tables the other way round (that is, table *GEO_REGIONS* as the left table), just change the spatial predicate to *Contains*.

a) In the *Join_1* node, to create the join, drag column *GEO_CUSTOMERS.CUSTOMER_LOC* to column *GEO_REGIONS.REGION_LOC*.

b) To edit the join properties, double-click the new connector.

c) In the *Predicate* dropdown list of the *Spatial Properties* tab, select *Within*.

d) Choose *OK*.

4. Save, Validate, and Activate the view and preview the data.

Check that the total sales for region *NSAM* are consistent with the amount you retrieved in the previous task.

a) Choose *Save and Validate*.

b) Choose *Save and Activate*.

c) Choose *Data Preview*.

d) Select the *Raw data* tab.

e) Check that the data of the calculation view is consistent.

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

LESSON SUMMARY

You should now be able to:

- Work with Geospatial Data
- Query geographical data using SQL

Unit 5
Lesson 5

Developing Predictive Models

LESSON OVERVIEW

Successful organizations don't just review historical information, they also look forward. They do this to gain insight that helps them to take advantage of opportunities and also avoid risk. In this lesson, we will get you started on predictive modeling with SAP HANA using the Predictive Analysis Library (PAL) functions.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Develop a predictive model using SAP HANA Predictive Analysis Library

Developing a Predictive Model using SAP HANA PAL

Predictive analytics is an area of data mining that deals with extracting information from data and using it to predict trends and behavior patterns. Often the unknown event of interest is in the future, but predictive analytics can be applied to any type of unknown, if it's in the past, present, or future.

Predictive analytics are gaining momentum in business applications as organizations shift from rearward facing analysis to forward facing analysis. There are limitless examples of where predictive analytics can be applied in any organization and in any industry to help identify opportunities or avoid risk.

Healthcare
Predict likelihood of disease to begin early treatment; identify clinical trial outcomes.

CRM Sales
Identify revenue forecast based on customer opportunities and pipelines execution.

Banking
Identify key behaviours of customers likely to leave the bank; improve credit risk analysis.

Utilities
Forecast demand and usage for seasonal operations; provide anticipated resources.

CRM Marketing
Identify potential leads among existing customers and intelligently market to them based on individual preferences and histories

Government
Predict community movement and trends that affect taxing districts; anticipate revenue.

Retail
Intelligent selection of store locations based on demographics; inventory planning.

Telco
Forecast demand on system load for capacity planning and customer scale.

Figure 313: Predictive Analysis Examples

Application Function Library

SAP have developed a library of functions called the Application Function library (AFL). This library contains two sub-libraries filled with pre-delivered, SAP developed functions:

- Business Function Library (BFL)

  Predefined parameter driven functions that encapsulate common business functions such as depreciation, days outstanding. These reuseable functions can be called from procedures and push down the processing of complex and data intense functions to the SAP HANA layer. So the main benefits are better productivity from the developer who does not have to code the business function, and great performance.

- Predictive Analysis Library (PAL)

  Predefined parameter driven functions that can be called from within procedures to perform analytic algorithms. PAL includes well known predictive analysis algorithms organized into nine data-mining categories:

  - Clustering
  - Classification
  - Regression
  - Association
  - Preprocessing
  - Statistics
  - Social Network Analysis
  - Time Series
  - Miscellaneous

  We will focus on PAL functions in this lesson.

Predictive Analysis Algorithms

PAL functions are written in C++ and can be called from SAP HANA procedures. You can use tables or views or output from other procedures as the data source to the predictive analysis. The results can be passed to tables or as inputs to other procedures.

Although many of the PAL functions call complex algorithms, it is not necessary to understand in detail the inner working of each algorithm and the math they contain. Many of the algorithms are well known, industry standard and publicly available and so those people who are experienced in this field will recognize many of them. SAP provide detailed documentation that describes each algorithm and its use cases. You can find the documentation at `help.sap.com`.

- **Association Analysis**
  - Apriori
  - FP-Growth
  - K-Optimal Rule Discovery (KORD)
- **Classification Analysis**
  - Area Under Curve (AUC)
  - Back Propagation Neural Network
  - C4.5 Decision Tree
  - CART Decision Tree
  - CHAID Decision Tree
  - Confusion Matrix
  - KNN
  - Logistic Regression (with Elastic Net Regularization)
  - Multi-Class Logistic Regression
  - ContentNaive Bayes
  - Parameter Selection and Model Evaluation (PSME)
  - Predict with Tree Model
  - Random Forests
  - Support Vector Machine
  - Incremental Classification on SAP HANA Smart Data Streaming
- **Regression**
  - Bi-Variate Geometric Regression
  - Bi-Variate Natural Logarithmic Regression
  - Exponential Regression
  - Multiple Linear Regression
  - Polynomial Regression
- **Cluster Analysis**
  - Affinity Propagation
  - Agglomerate Hierarchical Clustering
  - Anomaly Detection
  - Cluster Assignment
  - DBSCAN
  - Gaussian Mixture Model (GMM)
  - K-Means
  - K-Medians
  - K-Medoids
  - LDA Estimation and Inference
  - Self-Organizing Maps
  - Slight Silhouette

Figure 314: Predictive Analysis Algorithms (1)

- **Time Series Analysis**
  - ARIMA
  - Brown Exponential Smoothing
  - Croston's Method
  - Forecast Accuracy Measures
  - Forecast Smoothing
  - Linear Regression with Damped Trend and Seasonal Adjust
  - Single Exponential Smoothing
  - Double Exponential Smoothing
  - Triple Exponential Smoothing
  - Seasonality Test
  - Trend Test
  - White Noise Test
- **Social Network Analysis**
  - Link Prediction
- **Data Preparation**
  - Binning
  - Binning Assignment
  - Convert Category Type to Binary Vector
  - Inter-Quartile Range Test
  - Partition
  - Posterior Scaling
  - Principal Component Analysis (PCA)
  - Random Distribution Sampling
  - Sampling
  - Scaling Range
  - Substitute Missing Values
- **Statistics Functions**
  - Chi-Squared Test for Fitness
  - Chi-Squared Test for Independent
  - Cumulative Distribution Function
  - Distribution Fitting
  - Grubbs' Test
  - Kaplan-Meier Survival Analysis
  - Multivariate Statistics
  - Quantile Function
  - Univariate Statistics
  - Variance Equal Test
- **Other**
  - ABC Analysis
  - Weighted Score Table

Figure 315: Predictive Analysis Algorithms (2)

PAL Installation

To get started with PAL, you must install the Application Function Library (AFL) which includes PAL (and also BFL). You must also ensure that the script server is started and that you have been assigned the role **`AFL_SYS_AFL_AFLPAL_EXECUTE`**.

Refer to the AFL installation guide at `help.sap.com` for more details.

Note:
To confirm that the PAL functions were installed successfully, you can check the following three public views are available:

- sys.afl_areas
- sys.afl_packages
- sys.afl_functions

To check the views, run the following SQL statements:

- SELECT * FROM "SYS"."AFL_AREAS" WHERE AREA_NAME = 'AFLPAL';
- SELECT * FROM "SYS"."AFL_PACKAGES" WHERE AREA_NAME = 'AFLPAL';
- SELECT * FROM "SYS"."AFL_FUNCTIONS" WHERE AREA_NAME = 'AFLPAL';

The result will tell you whether the PAL functions were successfully installed on your system.

The SAP HANA Application Function Modeler (AFM) is a general purpose graphical modeler embedded in the SAP HANA studio. It is used to develop a variety of different models, these include ETL flows (for use with SDI / SDQ), data streaming flows (SDS) and also flows that call business functions (from BFL) and also Predictive functions (from PAL).

Flowgraph Example

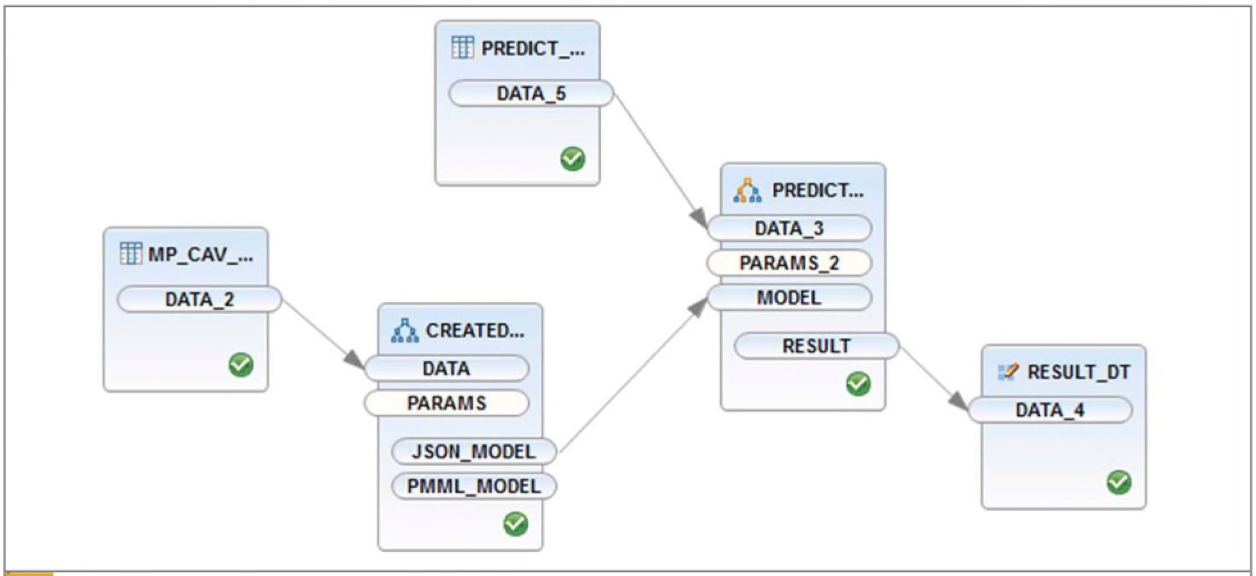


Figure 316: Flowgraph Example

With the AFM, you can easily add PAL function nodes to your flow, specify its modeling parameters, define input sources and output targets, and generate the procedure, all without writing any SQLScript code. You can also execute the procedure to get the output result of the function, and save the auto-generated SQLScript code for future use.

To get started you need to create an SAP HANA Project. This can be done using the *Project Explorer* view in the *Development Perspective*. Within your Project, you then create a flowgraph type file where you develop a flow that should include inputs, one or more functions from PAL, and outputs. So make sure you select this type of node when choosing a file type so that the correct editor is launched when you open the file.

> Note:
> It is easy to identify a predictive mode in the Project Explorer. Simply look for files with the extension .hdbflowgraph.

There are many PAL functions to choose from. In this course, we will show how easy it is to develop a predictive model that uses a decision tree.

A decision tree produces an outcome based on input variables. The more input variables you can provide to the decision tree, the higher the quality of prediction. A decision tree is one of a class of predictive models that need to be first trained before they can be deployed. Training a model involves passing known outcomes into the model so that the model is able to figure out what were the most influential variables that caused the various historical outcomes. The outcomes are often binary, in other words, a simple 'delayed' or 'on-time' decision. But decision trees can also provide multiple outcomes, for example, 'high', 'medium' and 'low'.

Unit 5
Exercise 25

Develop a Predictive Model

Business Example

You want to identify flights that are likely to be delayed. You have collected historical flight data and you will use this data to first train a predictive model. You will then use the trained model to predict flight delays on new data.

In this exercise, when values include ##, replace the characters with the number your instructor assigned to you.

Task 1: Create a Repository Workspace and a Project

To work with the SAP HANA Predictive Analysis Library, you need to work in the *SAP HANA Development* perspective.

You must create a local workspace to store "physically" the objects you will create, as well as a project to organize them.

1. Switch to the *SAP HANA Development* perspective and create a repository workspace using data in the following table:

Field	Value
Workspace Name	**WS_##**

2. Create a project and assign it to the workspace you created in the previous step. Use the data in the following table:

Field	Value
Project Name	**PAL_PROJ_##**

Task 2: Create a Predictive Model

1. Create a flowgraph development object in your project using data from the following table:

Field	Value
File name	**PAL_FG_##**

2. Set the target schema for the flow to your own *STUDENT##* schema.
3. Add the PAL classification function *Decision Tree C4.5* to your canvas.
4. Drag the column view *mp/CAV_TRAINDT* from the *_SYS_BIC* schema to the left side of the canvas. This will provide the input training data to the function.
5. Make a connection between *mp/CAV_TRAINDT* and the *DATA* node of the PAL function. Do not change the default column mapping.

6. In the *Mappings* properties of the connector, rearrange the *Target* columns so that they follow this order:
   - DAYOFWEEK
   - TIMEBLK
   - AIRLINE
   - DELAY_FLAG
7. Add the PAL classification function *Decision Tree (Predict)* to the canvas on the right side.
8. Create a connection between *JSON_MODEL* and *Model*.
9. Add the table *RESULT_DT* from your schema *STUDENT##* to the right side of the canvas as a data sink. This table will be used to store the results of the predictions.
10. Create a connection between *RESULT* and *DATA_4* and create the mappings as shown in the table:

Source	Target
ID	ID
PREDICTION	DELAY_FLAG

11. Add the table *PREDICT_WITH_DT* from the schema *FStats* to the top centre of the canvas as a data source.

    This table will be used to provide the new transactions on which flight delay predictions are required.
12. Create a connection between *DATA_5* and *DATA_3* and leave the column settings as default.
13. In the *Mappings* properties of the connector, move the *Target* columns Up/Down so that they follow this order:
    - ID
    - DAYOFWEEK
    - TIMEBLK
    - AIRLINE
14. Save the predictive analysis.
15. Activate the predictive analysis.

Task 3: Analyze the Available Data and Perform Predictions on New Data

You are now ready to analyze the delays for past flights. Based on this analysis, you will then predict the delays of upcoming flights.

1. Execute the analysis in the SQL Editor.
2. Observe the predictions that are found in the *RESULT_DT* table in your own schema *STUDENT##*.

Note:
If you need to re-run the predictive model, you need to delete all the entries in the table *RESULT_DT* of your schema *STUDENT##* to prevent a run-time error.

Unit 5
Solution 25

Develop a Predictive Model

Business Example

You want to identify flights that are likely to be delayed. You have collected historical flight data and you will use this data to first train a predictive model. You will then use the trained model to predict flight delays on new data.

In this exercise, when values include ##, replace the characters with the number your instructor assigned to you.

Task 1: Create a Repository Workspace and a Project

To work with the SAP HANA Predictive Analysis Library, you need to work in the *SAP HANA Development* perspective.

You must create a local workspace to store "physically" the objects you will create, as well as a project to organize them.

1. Switch to the *SAP HANA Development* perspective and create a repository workspace using data in the following table:

Field	Value
Workspace Name	**WS_##**

   a) Choose *Window* → *Perspective* → *Open Perspective* → *SAP HANA Development*

   b) Select the tab *Repositories* and choose *Create Repository Workspace*.

   c) Deselect the *Use Default Workspace* property.

   d) At the dialog box, enter the data from the table.

   e) Choose *Finish*.

2. Create a project and assign it to the workspace you created in the previous step. Use the data in the following table:

Field	Value
Project Name	**PAL_PROJ_##**

   a) Select the *Project Explorer* tab, and from the context menu, choose *New* → *Other*.

   b) In the dialog box, select *SAP HANA* → *Application Development* → *XS Project* and choose *Next*.

   c) Enter the data from the table and choose *Next*.

d) Assign your new project to the workspace you created in the previous step by highlighting your workspace, then choose *Next*.

e) To complete the project setup, choose *Finish*.

Task 2: Create a Predictive Model

1. Create a flowgraph development object in your project using data from the following table:

Field	Value
File name	**PAL_FG_##**

   a) From the context menu of the *Project Explorer* tab, choose *New* → *Other*.

   b) In the dialog box, choose *SAP HANA* → *Database Development* → *Flowgraph Model* and choose *Next*.

   c) Ensure your project is selected and enter the data from the table.

   d) Choose *Finish*.
   The flowgraph object is saved and is automatically opened in the editor.

2. Set the target schema for the flow to your own *STUDENT##* schema.

   a) Choose the canvas of the editor and on the *General* tab of properties set the schema to your own **STUDENT##**.

3. Add the PAL classification function *Decision Tree C4.5* to your canvas.

   a) Expand the palette *Predictive Analysis Library* and then expand the section *Classification / Regression*.

   b) Drag the PAL function *Decision Tree C4.5* to the centre of the canvas.

4. Drag the column view *mp/CAV_TRAINDT* from the *_SYS_BIC* schema to the left side of the canvas. This will provide the input training data to the function.

   a) Under the *Project Explorer* on the left side of the screen, expand *SAP HANA System Library*.

   b) Expand the node *Catalog*.

   c) Expand the node *_SYS_BIC*.

   d) Expand the node *Column Views*.

   e) Scroll until you locate the column view *mp/CAV_TRAINDT* and drag this to the left side of the canvas.

5. Make a connection between *mp/CAV_TRAINDT* and the *DATA* node of the PAL function. Do not change the default column mapping.

   a) Hover the mouse over the data node of the column view until the arrow icon appears. Then draw a line from the arrow icon to the data node of the PAL function.

   b) At the mapping dialog, choose *Finish*.

6. In the *Mappings* properties of the connector, rearrange the *Target* columns so that they follow this order:

- DAYOFWEEK
- TIMEBLK
- AIRLINE
- DELAY_FLAG

a) Select the connector joining DATA_2 and DATA.

b) In the *Mappings of the Properties* tab, move the target columns so that they match the above column order, using the Up/Down buttons.

7. Add the PAL classification function *Decision Tree (Predict)* to the canvas on the right side.

   a) Expand the palette *Predictive Analysis Library* and then expand the section *Classification / Regression*.

   b) Drag the PAL function *Decision Tree (Predict)* to the right side of the canvas.

8. Create a connection between *JSON_MODEL* and *Model*.

   a) Hover the mouse over *JSON_MODEL* until an arrow appears.

   b) Draw a line from the arrow icon to *Model*.

9. Add the table *RESULT_DT* from your schema *STUDENT##* to the right side of the canvas as a data sink. This table will be used to store the results of the predictions.

   a) Under the *Project Explorer* on the left side of the screen, expand *SAP HANA System Library*.

   b) Expand the node *Catalog*.

   c) Expand the node *STUDENT##*.

   d) Expand the node *Tables*.

   e) Drag the table *RESULT_DT* to the right side of the canvas and when prompted, choose *Data Sink*.

10. Create a connection between *RESULT* and *DATA_4* and create the mappings as shown in the table:

Source	Target
ID	ID
PREDICTION	DELAY_FLAG

   a) Hover the mouse over *RESULT* until an arrow appears.

   b) Draw a line from the arrow icon to *DATA_4*.

   c) Select the new line you just created by clicking anywhere on it.

   d) In the *Properties* view, select the *Mappings* tab.

   e) Draw a line between the column *PREDICTION* and *DELAY_FLAG*.

11. Add the table *PREDICT_WITH_DT* from the schema *FStats* to the top centre of the canvas as a data source.

This table will be used to provide the new transactions on which flight delay predictions are required.

a) Under the *Project Explorer* on the left side of the screen, expand *SAP HANA System Library*.

b) Expand the node *Catalog*.

c) Expand the node *FStats*.

d) Expand the node *Tables*.

e) Drag the table *PREDICT_WITH_DT* to the top centre of the canvas and when prompted, choose *Data Source*.

12. Create a connection between *DATA_5* and *DATA_3* and leave the column settings as default.

a) Hover the mouse over *DATA_5* until an arrow appears.

b) Draw a line from the arrow icon to *DATA_3*.

c) Choose *Finish* to close the dialog.

13. In the *Mappings* properties of the connector, move the *Target* columns Up/Down so that they follow this order:

- ID
- DAYOFWEEK
- TIMEBLK
- AIRLINE

a) Select the connector joining DATA_5 and DATA_3

b) In the *Mappings of the Properties* tab, move the target columns so that they match the above column order, using the Up/Down buttons.

14. Save the predictive analysis.

a) Choose *Save* in the toolbar.

15. Activate the predictive analysis.

a) Choose *Activate* in the toolbar.

b) In the *Job Log* view, check that the activation job status is *Completed successfully*.

Task 3: Analyze the Available Data and Perform Predictions on New Data

You are now ready to analyze the delays for past flights. Based on this analysis, you will then predict the delays of upcoming flights.

1. Execute the analysis in the SQL Editor.

a) Select the option *Execute in SQL Editor* icon in the toolbar.

b) The SQL Console appears after a few seconds and by observing the messages at the bottom of the screen, you will observe the procedure has executed.

2. Observe the predictions that are found in the *RESULT_DT* table in your own schema *STUDENT##*.

> Note:
> If you need to re-run the predictive model, you need to delete all the entries in the table *RESULT_DT* of your schema *STUDENT##* to prevent a run-time error.

a) Under the *Project Explorer* on the left side of the screen, expand *SAP HANA System Library*.

b) Expand the *Catalog*.

c) Expand the node *STUDENT##*.

d) Expand the node *Tables*.

e) From the context menu of the table *RESULT_DT*, select *Open Data Preview* and you should find predictions for three flights. The predictions are indicated with 1 (true) or 0 (false).

LESSON SUMMARY

You should now be able to:

- Develop a predictive model using SAP HANA Predictive Analysis Library

Unit 5
Lesson 6

Working with SAP HANA Graph

LESSON OVERVIEW

This lesson provides an overview of SAP HANA Graph, which is a new feature of SAP HANA SPS12.

Graph processing enables you to make better use of your data by organizing it into nodes and relationships between nodes. This facilitates the visualization and in-depth analysis of your data, in particular with the help of specific algorithms.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe graph processing with SAP HANA

Graph Modeling

Graphs are used to model data that is best represented using a network. For example, logistics and transportation, utility networks, and social networks. The basic idea behind graph modeling is that it allows a modeler to easily define a series of entities (nodes) and link them in a network that represents how they relate to each other. Graph models can indicate flow direction between entities so that additional meaning can be added to the network and traces can be made. Imagine a complex supply chain mapped using a graph, where all manufacturers, suppliers, distributors, customers, and consumers are all represented with information stored along the connections. But why would you want to define such models? The benefit is that it is easy to develop applications that can traverse huge graphs at speed so you can ask questions such as:

How many hours has the product traveled between two specified points in the network?
Where are all the possible points of origin of this product?
Describe the entire journey of a product by listing all stop off points.

Graph processing allows us to discover hidden patterns and relationships in huge amounts of data and all in real time.

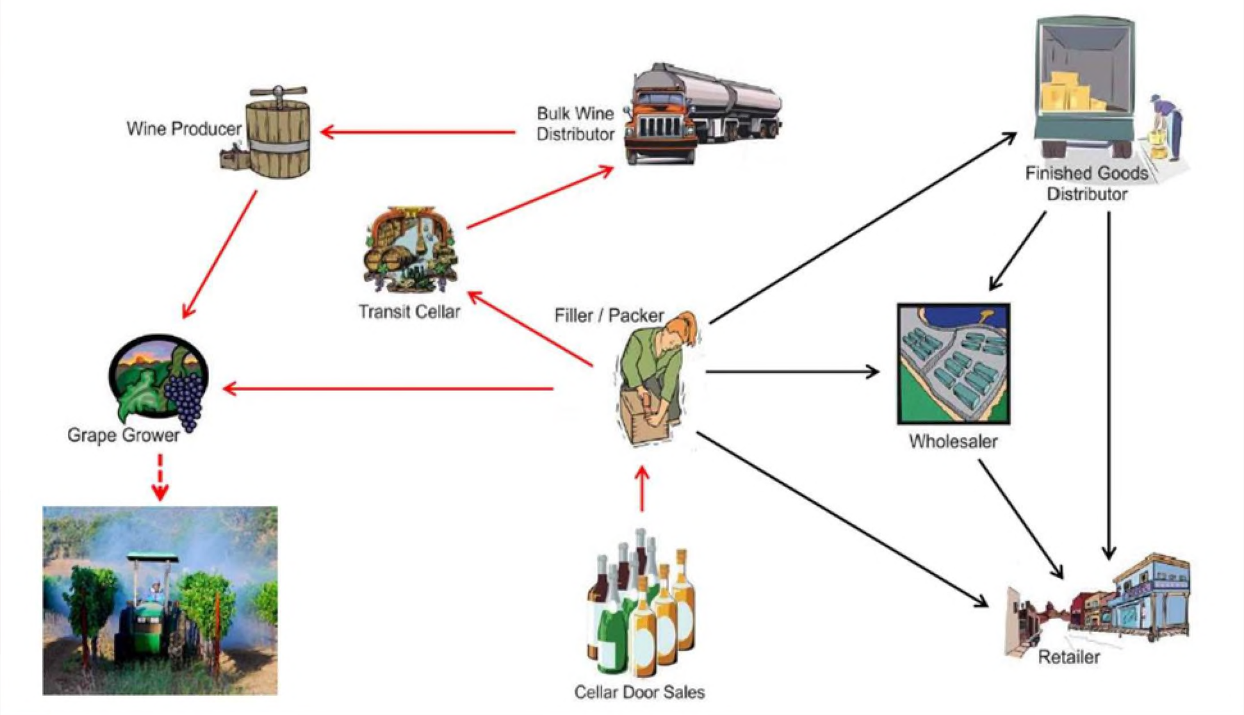


Figure 317: Example graph model

The example we use here is one that most people can relate to, but there are many other interesting examples such as:

- Medical — create a network of patients, conditions, treatments and outcomes for re-use in diagnosis and planning treatments of other patients.
- Social network — using popular social media portals, find your customers and their friends and friends of friends and likes and dislikes to create marketing opportunities

Although it is possible to use standard SQL data definitions and query syntax to create and process a similar model, it would be extremely complex both in the definition of the model and also the SQL needed for the querying of the graph. Plus processing times could be challenging. SAP HANA Graph provides tools for graph definition and language for graph processing to ensure that model development is more natural and simplified and the processing is flexible, and of course optimized for in-memory processing.

How is Graph Modeling Implemented with SAP HANA?

To get started let's study an example of a graph model based on Greek mythology.

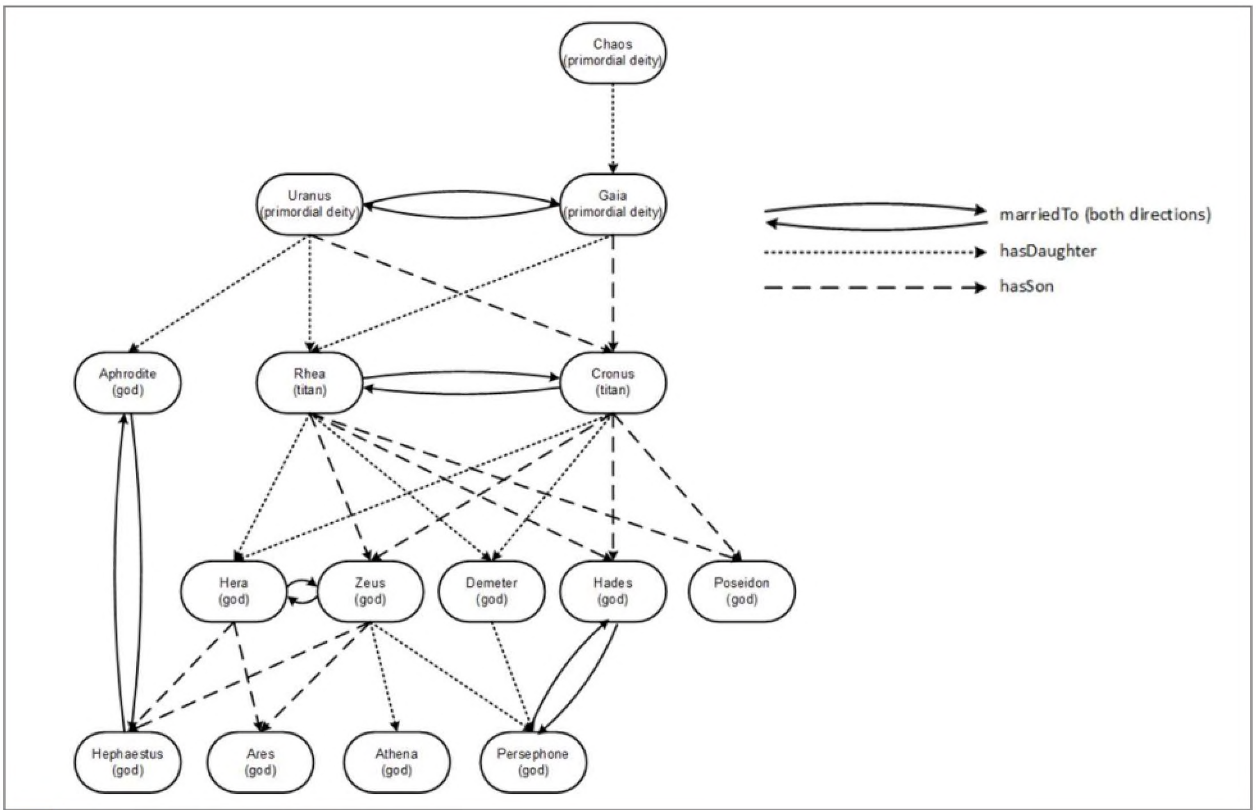


Figure 318: SAP HANA Graph Greek model

Note:
This example is taken from the standard SAP documentation where all the SQL code is available to copy and paste to create your very own Greek mythology graph model, just like this one.

Let's look at how we create this model in SAP HANA.

The key objects in a graph model are the vertices and edges. Vertices are stored in column tables and represent the nodes of the graph. In each vertex (singular for vertices) we can store multiple attributes. The edges are also stored in column tables and describe the lines between the members. With each edge, we can store relationship attributes such as distance, preference, strength, and so on. Edges describe the relationships between the vertices. It is possible to have multiple edges between two vertices. Edges can go in both directions between the vertices to represent the direction of the relationship. For example, a YouTube vlogger and a follower, father and daughter.

So to get started with SAP HANA Graph you need to:

1. Create and fill the vertices table and edge table with graph data. In case the data is already available in existing tables you can also create views over the tables so that the data is in the format required by graph modeling.

   The vertices table and edge table do not need to reside in the same schemas.

```
CREATE SCHEMA "GREEK_MYTHOLOGY";

CREATE COLUMN TABLE "GREEK_MYTHOLOGY"."MEMBERS" (
  "NAME" VARCHAR(100) PRIMARY KEY,
  "TYPE" VARCHAR(100),
  "RESIDENCE" VARCHAR(100)
```

Vertices

```
CREATE COLUMN TABLE "GREEK_MYTHOLOGY"."RELATIONSHIPS" (
  "KEY" INT UNIQUE NOT NULL,
  "SOURCE" VARCHAR(100) NOT NULL
    REFERENCES "GREEK_MYTHOLOGY"."MEMBERS" ("NAME")
    ON UPDATE CASCADE ON DELETE CASCADE,
  "TARGET" VARCHAR(100) NOT NULL
    REFERENCES "GREEK_MYTHOLOGY"."MEMBERS" ("NAME")
    ON UPDATE CASCADE ON DELETE CASCADE,
  "TYPE" VARCHAR(100)
```

Edges

Figure 319: SAP HANA Graph create vertice and edge tables

2. Create a Graph Workspace that refers to the vertices and edge tables. The Graph Workspace simply creates the metadata that defines the graph model and how the vertices and edge tables relate. No actual business data is stored in a Graph Workspace. After activation the resulting workspace will then appear in the system table SYS.GRAPH_WORKSPACES where all graph workspaces are listed.

Create a graph workspace

```
CREATE GRAPH WORKSPACE "GREEK_MYTHOLOGY"."GRAPH"
  EDGE TABLE "GREEK_MYTHOLOGY"."RELATIONSHIPS"
    SOURCE COLUMN "SOURCE"
    TARGET COLUMN "TARGET"
    KEY COLUMN "KEY"
  VERTEX TABLE "GREEK_MYTHOLOGY"."MEMBERS"
    KEY COLUMN "NAME";
```

Figure 320: SAP HANA Graph create workspace

3. Build queries over the workspace to ask questions about the members of the graph, such as distance, shortest path etc.

Queries can be built directly using SQLScript but SAP also supplies a tool called the SAP HANA Graph Viewer that is very easy to use and helps to quickly formulate queries over the workspace and generates the SQLScript that you can then copy and paste to your own applications.

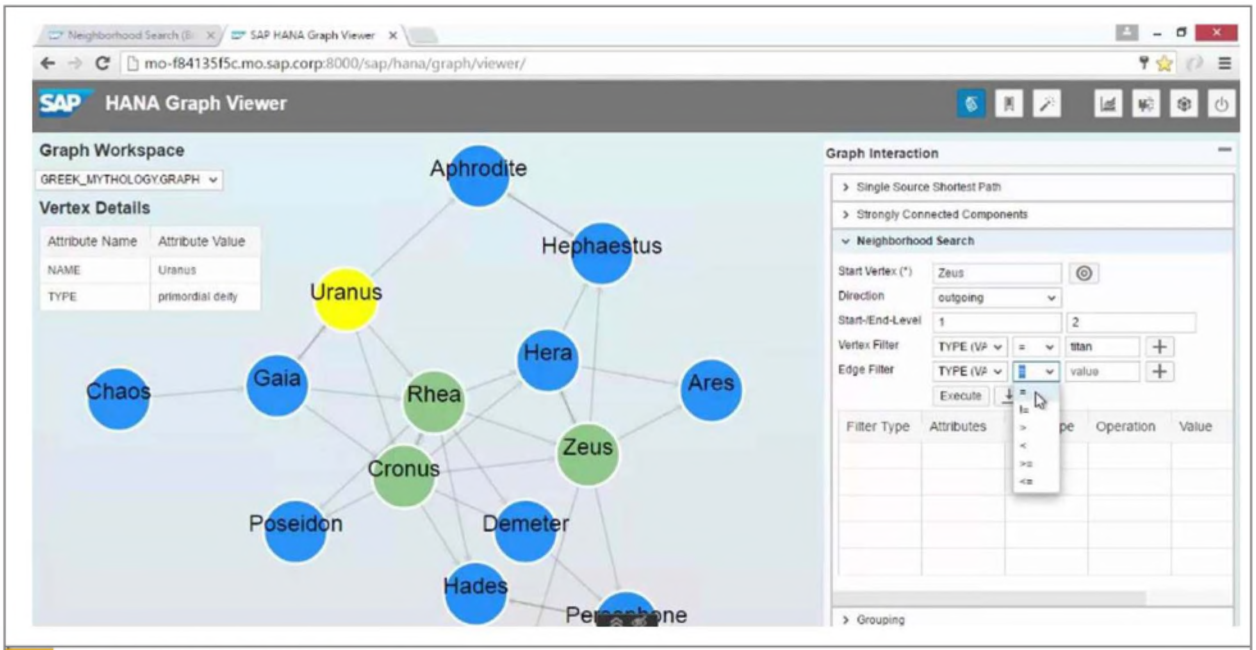


Figure 321: SAP HANA Graph Viewer

The Graph Viewer is a native SAP HANA application built with SAPUI5 and utilizes HANA XS. It provides a simple interface to interact with and visualize graph workspaces in SAP HANA. SAP HANA Graph Viewer is an additional tool for SAP HANA Graph and is not part of the basic installation of SAP HANA. It can be downloaded from the SAP Software Download Center.

Note:
There are no special security privileges needed to launch and use the Graph Viewer but you will need rights to the tables and objects being viewed.

SAP HANA Graph provides a native processing language for graph query and manipulation called WIPE (weakly structured information processing and exploration). It is a declarative language very similar to SQL and contains special keywords to allow easy graph based questions to be formulated such as, how far? how deep? what's the source? where is the end point? where is the strongest connection? what is the shortest path?

SAP HANA Graph utilizes a dedicated graph engine that works with the other data processing engines in SAP HANA. The graph engine is part of the standard SAP HANA installation.

SAP HANA Graph processing can be combined with all other types of SAP HANA data processing such as textual, spatial and predictive so that sophisticated and innovative applications can be developed on any data types.

LESSON SUMMARY

You should now be able to:

- Describe graph processing with SAP HANA

Learning Assessment

1. While accessing data through Information Access Services, Fuzzy Search queries are handled by the Text Processor.

   Determine whether this statement is true or false.

   - [ ] True
   - [ ] False

2. Identify the data types supported by the Fuzzy Search.

   Choose the correct answers.

   - [ ] A TEXT
   - [ ] B SHORT TEXT
   - [ ] C VARCHAR
   - [ ] D VARBINARY
   - [ ] E NVARCHAR
   - [ ] F DATE

3. Which of the following functions is used in the `WHERE` clause to perform a Full Text Search?

   Choose the correct answer.

   - [ ] A CONTAINS()
   - [ ] B CONSISTS()
   - [ ] C HIGHLIGHTED()
   - [ ] D PROJECTION()

4. It is possible to create database queries to answer questions like “Which of my customers are close to a certain store?” by utilizing geospatial features.

   Determine whether this statement is true or false.

   - [ ] True
   - [ ] False

5. Which of the following are supported by the spatial features that form part of the SAP HANA architecture?

   Choose the correct answers.

   - [ ] A 2D – Vector Types
   - [ ] B Points, line-strings, polygons, compound polygons
   - [ ] C 3D – Vector Types
   - [ ] D Hexagons and compound hexagons

6. What do you use to define a predictive analysis model?

   Choose the correct answer.

   - [ ] A A decision tree
   - [ ] B A flowgraph
   - [ ] C A calculation view
   - [ ] D An attribute view

7. A decision tree must be trained before it can be deployed.

   Determine whether this statement is true or false.

   - [ ] True
   - [ ] False

Learning Assessment - Answers

1. While accessing data through Information Access Services, Fuzzy Search queries are handled by the Text Processor.

   Determine whether this statement is true or false.

   [ ] True

   [X] False

2. Identify the data types supported by the Fuzzy Search.

   Choose the correct answers.

   [X] A TEXT

   [X] B SHORT TEXT

   [X] C VARCHAR

   [ ] D VARBINARY

   [X] E NVARCHAR

   [X] F DATE

3. Which of the following functions is used in the `WHERE` clause to perform a Full Text Search?

   Choose the correct answer.

   [X] A CONTAINS()

   [ ] B CONSISTS()

   [ ] C HIGHLIGHTED()

   [ ] D PROJECTION()

4. It is possible to create database queries to answer questions like “Which of my customers are close to a certain store?” by utilizing geospatial features.

   Determine whether this statement is true or false.

   - [X] True
   - [ ] False

5. Which of the following are supported by the spatial features that form part of the SAP HANA architecture?

   Choose the correct answers.

   - [X] A 2D – Vector Types
   - [X] B Points, line-strings, polygons, compound polygons
   - [X] C 3D – Vector Types
   - [ ] D Hexagons and compound hexagons

6. What do you use to define a predictive analysis model?

   Choose the correct answer.

   - [ ] A A decision tree
   - [X] B A flowgraph
   - [ ] C A calculation view
   - [ ] D An attribute view

7. A decision tree must be trained before it can be deployed.

   Determine whether this statement is true or false.

   - [X] True
   - [ ] False

UNIT 6 Management and Administration of Models

UNIT OBJECTIVES

- Process and analyze models
- Compare different versions of an Information Model
- Check model references
- Generate auto documentation
- Execute a Performance Analysis of an Information Model
- Manage modeling content
- Define package-specific authoring schema
- Transport information models between two systems
- Move and copy information models within a system
- Take over an information model edited by another user
- Translate metadata texts
- Convert Attribute and Analytic Views into Calculation Views
- Convert Scripted Calculation Views into Table Functions

Unit 6
Lesson 1

Processing and Analyzing Information Models

LESSON OVERVIEW

In this lesson, you will learn about several features related to the lifecycle of an information model, such as validation, version comparison, documentation, and performance analysis.

Business Example

During the development phase of your project, you are involved in troubleshooting of some *AnalyticViews*.

You want to use different types of model validation rules to narrow down the list of issues and perform a more specific analysis.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Process and analyze models
- Compare different versions of an Information Model
- Check model references
- Generate auto documentation
- Execute a Performance Analysis of an Information Model

Validating and Activating Information Models

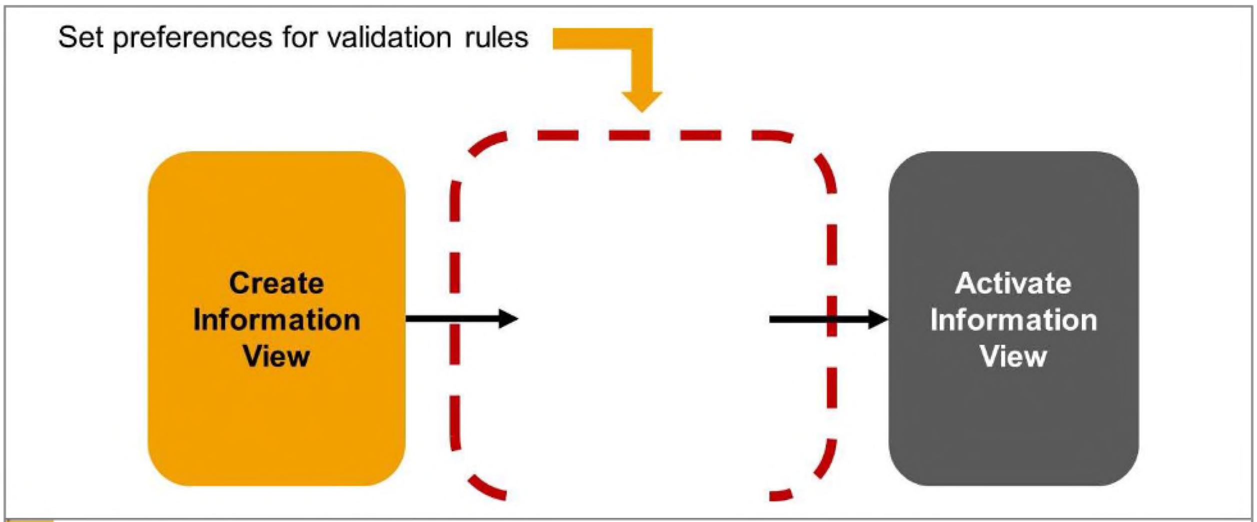


Figure 322: Validating Models

When you validate a Data Model, you check that its definition is consistent according to a number of validation rules that are defined in the preferences of SAP HANA studio.

Depending on how critical the check is for the validity of the models, the following types of rules apply:

- Warning

  Rules with a *Warning* severity do not prevent the activity of the information model. They help the Information Model designer to apply best practices (sometimes, related to the use of specific development languages or client tools) and consider design options that might lead to data inconsistency when reporting on the model.

- Error

  Rules with an *Error* severity are the most critical. In most cases, a model cannot be validated if one of these rules is not met.

> Note:
> Some of the validation rules are focussing on information model performance optimization. They are covered at the end of this lesson.

Setting Preferences for Validation Rules (1)

- In SAP HANA STUDIO, choose *Windows* → *Preferences*
- Select *SAP HANA* → *Modeler* → *Validation Rules*

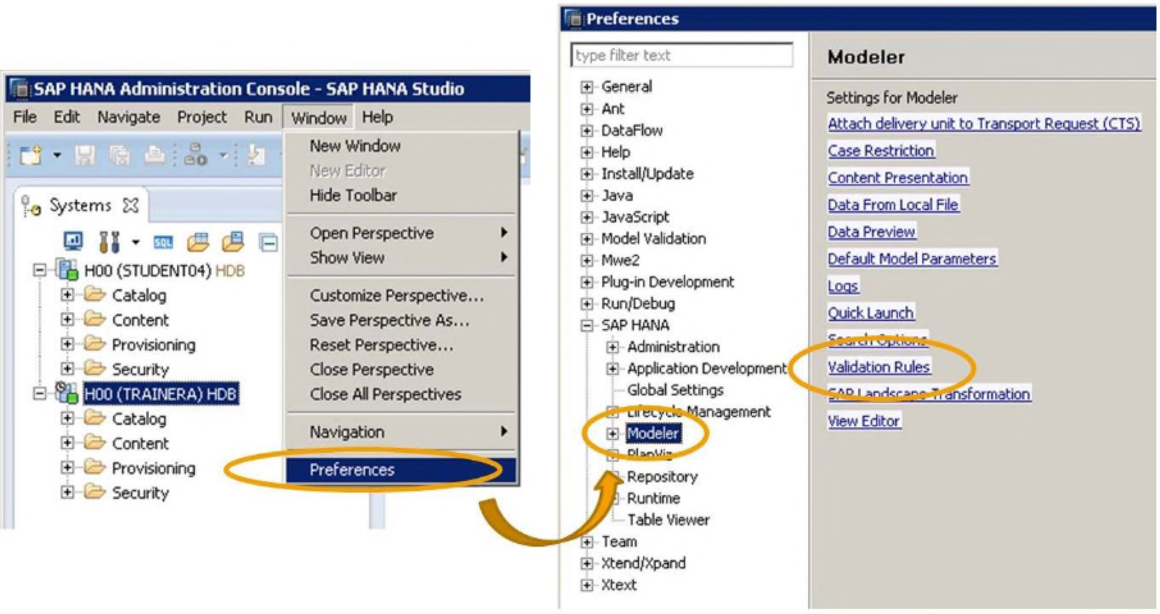


Figure 323: Setting Preferences for Validation Rules (1)

To visualize the complete set of validation rules, choose *Windows* → *Preferences*, and browse the preference tree to *SAP HANA* → *Modeler* → *Validation Rules*.

Setting Preferences for Validation Rules (2)

- You can select precisely which rules are applied during the validation of the Information Objects

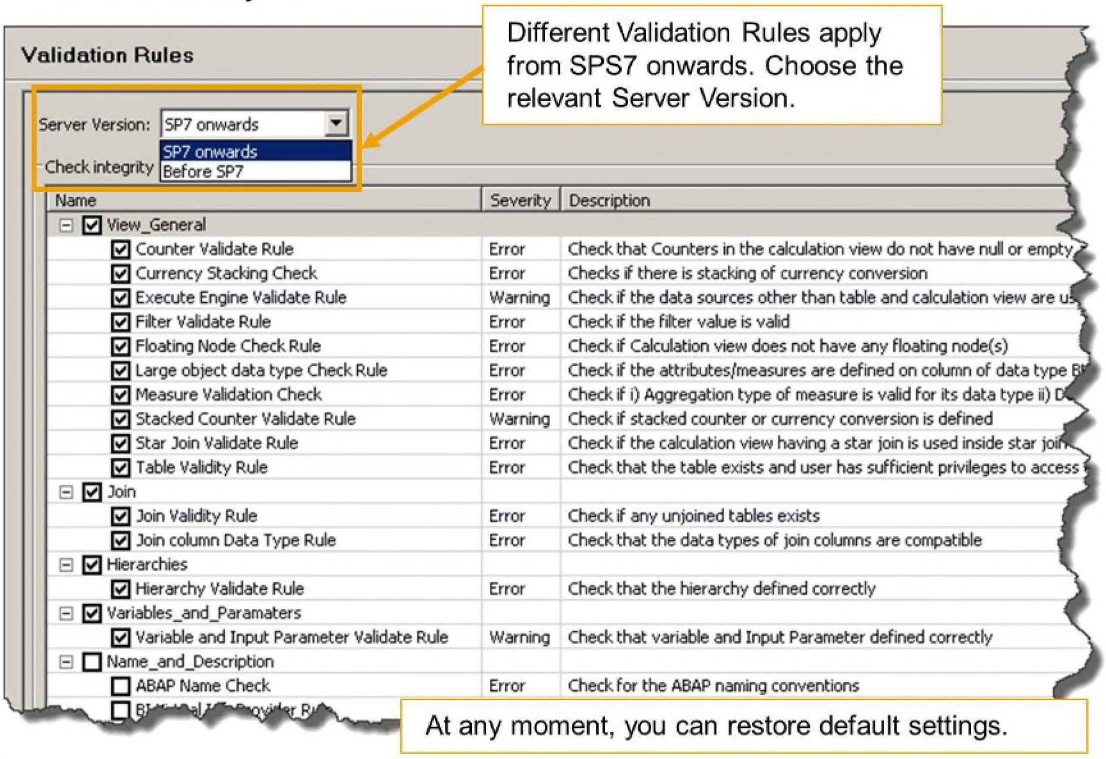


Figure 324: Setting Preferences for Validation Rules (2)

Depending on your needs, you can decide to disable some of the validation rules.

For example, the rule *Content → Description Check* checks that the label of a view or column is not equal to its name. If this control is not relevant in your case, you can deactivate this validation rule.

> Note:
> When you execute the validation of information with one or several validation rules disabled, you never get the job status, *Completed Successfully*.
>
> If the validation actually succeeded, the job status is *Completed with Warning*, to remind you that one or several rules were disabled during validation.

Executing Validation Rules

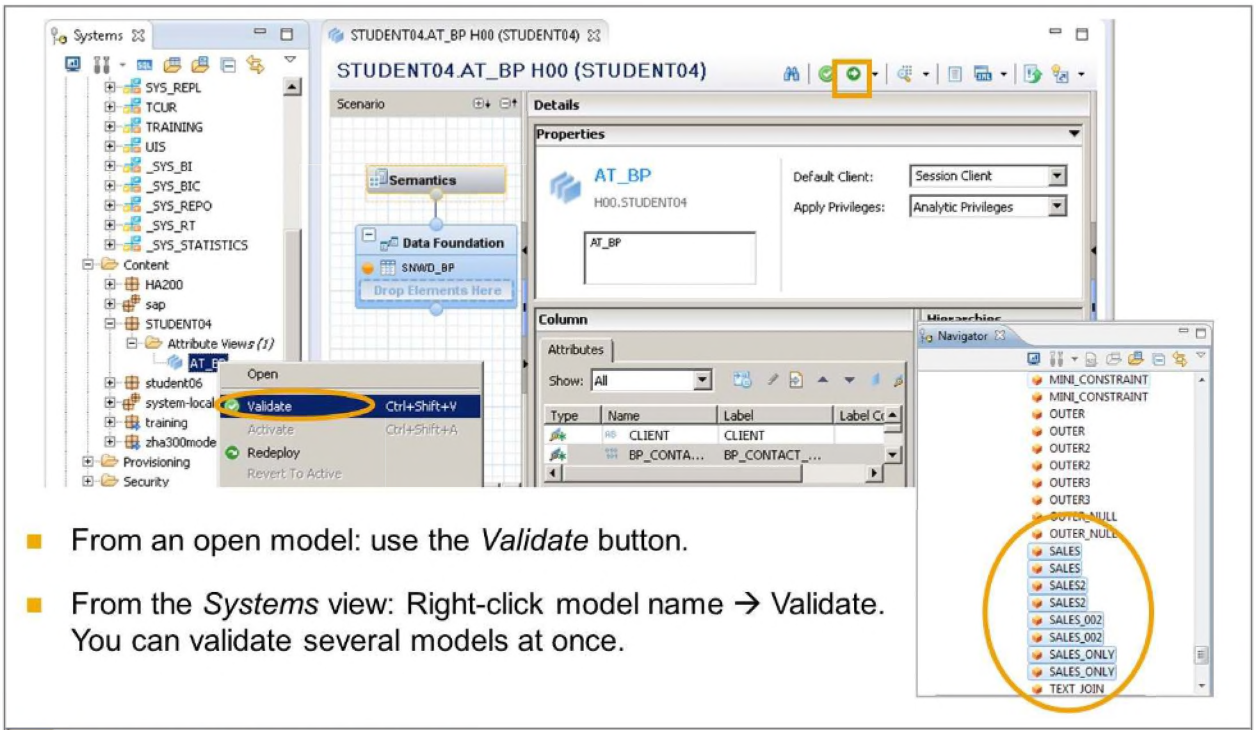


Figure 325: Executing Validation Rules

The ways to trigger the validation of information models includes the following:

- From the opened information model, choose *(Save and Validate)*.
- From the Systems view, select one or several information models and choose *(Validate)*, or press Ctrl+Shift+V.

Analyzing Validation Log

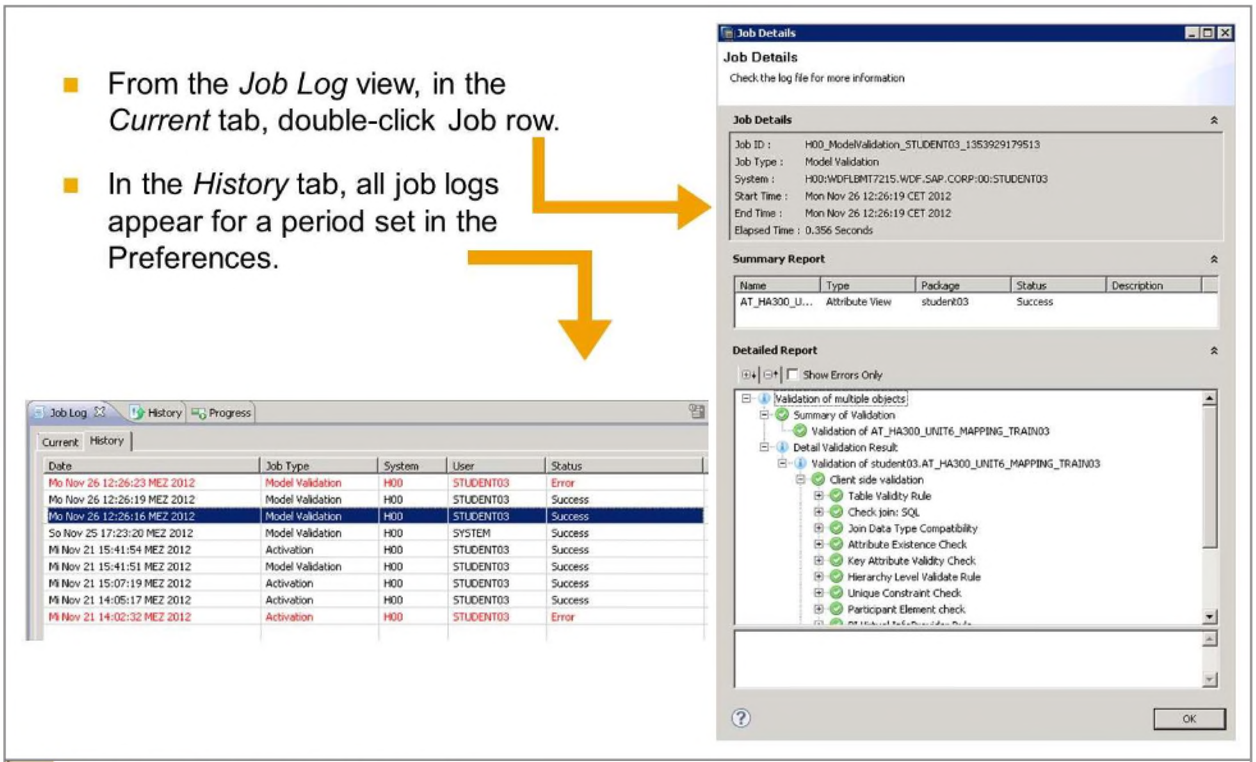


Figure 326: Analyzing Validation Log

The Validation Log provides a list of all the rules that generated a warning or an error during the validation of the Information Model.

It also provides a list of the rules that were not checked due to the Validation Rules settings in the SAP HANA Studio preferences.

Activating Information Models

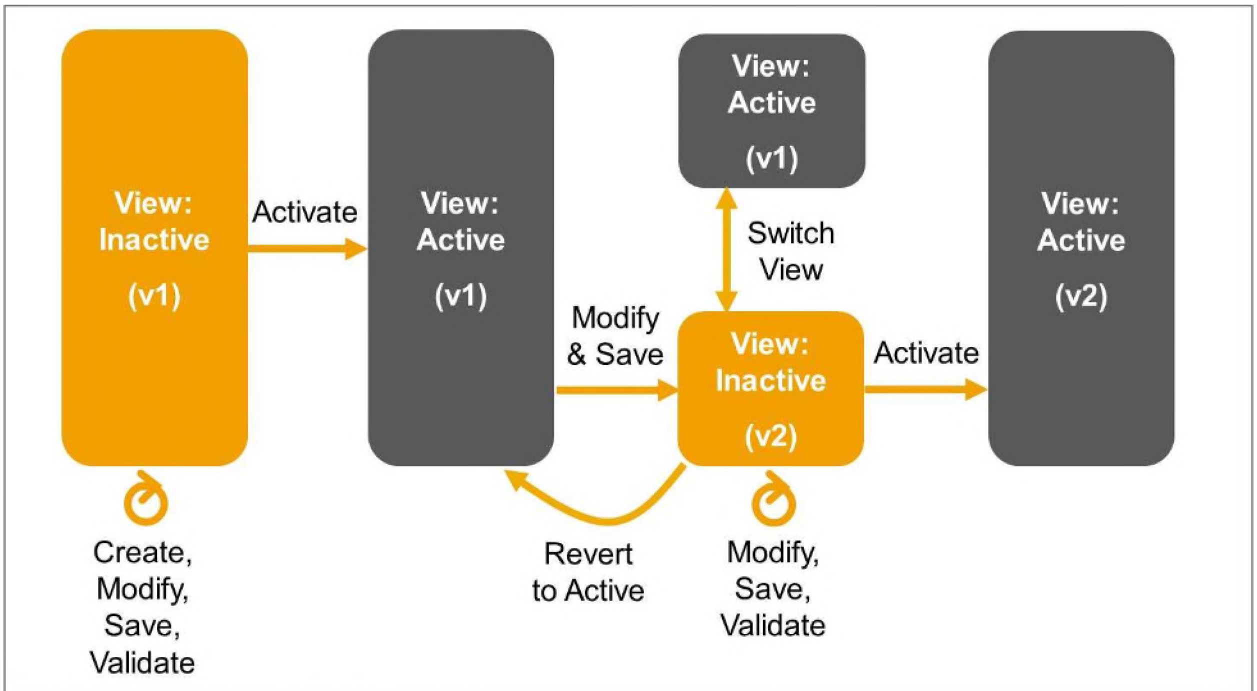


Figure 327: Information Model Lifecycle

When you create an information model, you work on an inactive version of the model that is stored in your local repository.

The Activation of an information model is the process that creates the column views of your model in the database catalog, in the schema _*SYS_BIC*, and allows the model consumption by reporting tools. These column views are the runtime version of the model.

> Note:
> The activated version of the object is generated in the schema _*SYS_BIC* only if the validation succeeds.

When the model has been activated for the first time and if you modify and save it, there are two versions of the model:

- The Inactive Version

  The inactive version is the one you are modifying. You can save it and activate it.

- The Active Version

  The Active version is still available in SAP HANA, that is, the column views in the schema _*SYS_BIC* still exist and can be used for reporting. You can easily switch between versions, in particular if you want to compare the active and inactive versions.

Object Versions - Comparing versions

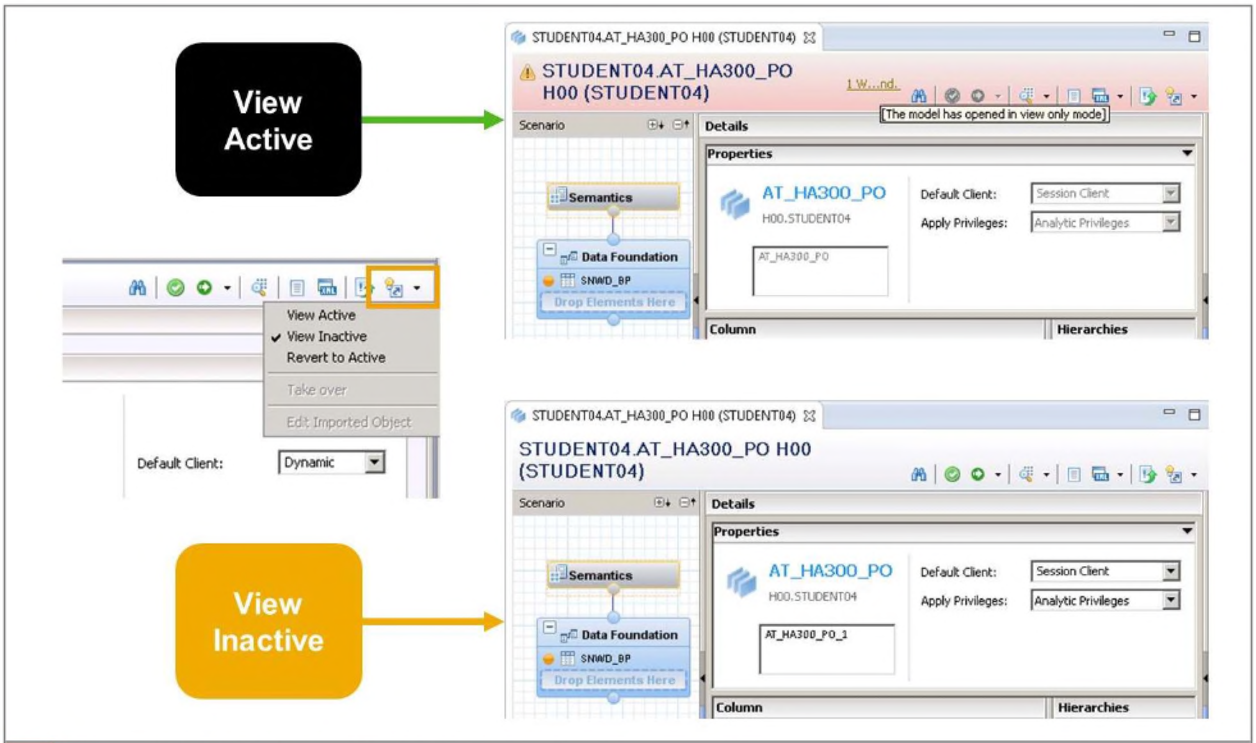


Figure 328: Object Versions - Comparing versions

SAP HANA allows you to display both the *Active* and *Inactive* versions of a view in two different screens when they exist at the same time.

Note:
When an active and inactive version of a view exists in the system, the *Active* view is in *Read-only* mode. In other words, the system does not allow you to have two different inactive views.

If needed, you can revert to the active version, which means that all the changes made were discarded when the view was activated for the last time.

The SAP HANA Development perspective allows you to revert to any of the local versions of an Information Model.

These local versions are stored on your local workspace when you edit them, and they stay in this location until you remove them or reset the workspace.

Further Details of SAP HANA Models

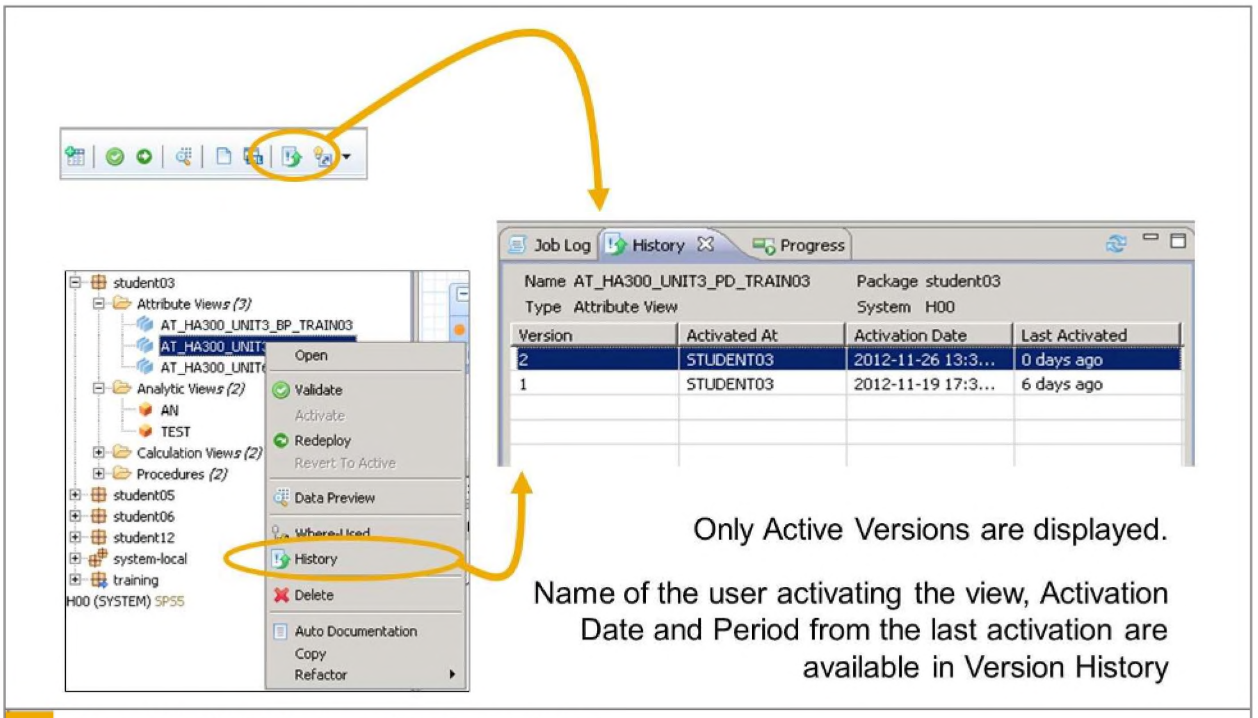


Figure 329: Object Versions - View Version History

It is possible to display the history of an information model.

The history includes one entry for each activated version, with the name of the user who activated the model and the date of activation.

Checking Model References (1)

- In the modeler, it is possible to check where an Information objects in used.
- This function is very helpful to study the impacts of changes in the data model.
- Right click the object, and choose « Where Used ».

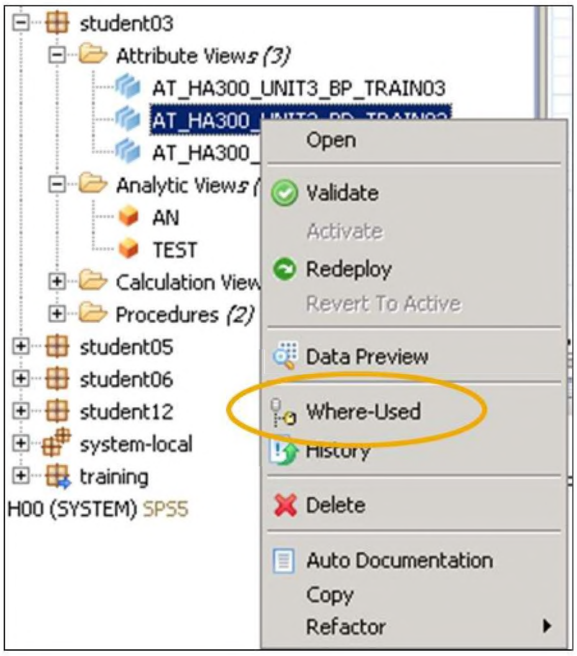


Figure 330: Checking Model References (1)

Checking Model References (2)

- The list displays the type, name and location (package) of each object that references the selected information model.
- You can also see, as a summary, the total number of objects in the list.

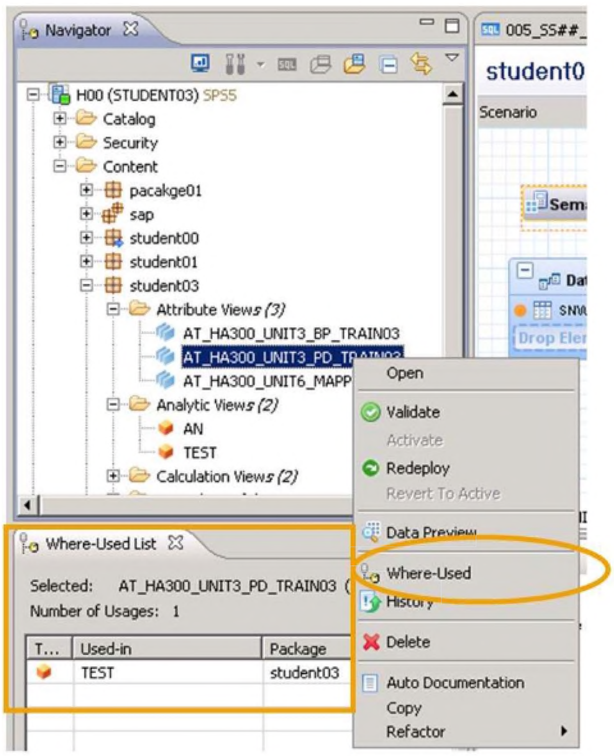


Figure 331: Checking Model References (2)

The *Where-Used* feature displays all the objects that reference a particular Information Model.

Auto Documentation

- It is possible to generate automatically documentation about the information models in SAP HANA. These documents can provide a list of all objects contained in a package, and the detailed design of selected models.
- Each information model gets its own Auto Documentation file (.pdf)
- You can generate Auto Documentation with a right click on an Information Model or directly with the button.



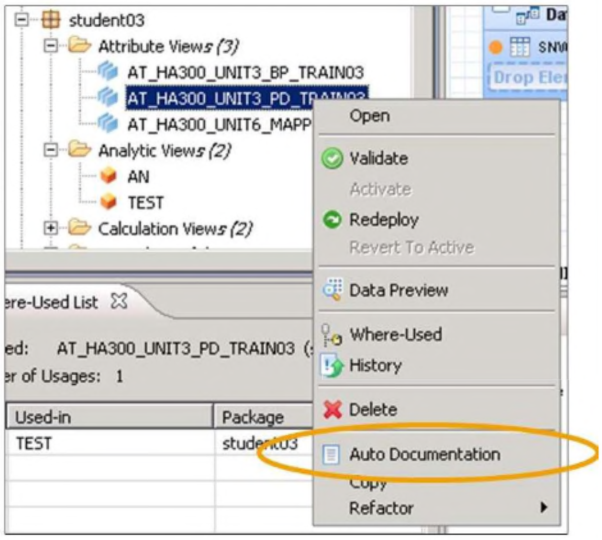


Figure 332: Auto Documentation

Auto Documentation generates the list of Information Models in a package, or the details about the design of one or several information models.

> Note:
> Not all content from Packages can be included in auto documentation. Only Information Models are included. This means you can include Attribute, Analytic, and Calculation views plus information about the structure of the packages. You cannot include procedures, decision tables, or analytic privileges in auto documentation.

The output consists of one or several .PDF files.

Auto Documentation - Select Document Type

- Two document types are available:
  - Model Details (display each particularity of an Information Objects)
  - Model List (display a list of each component of the package)

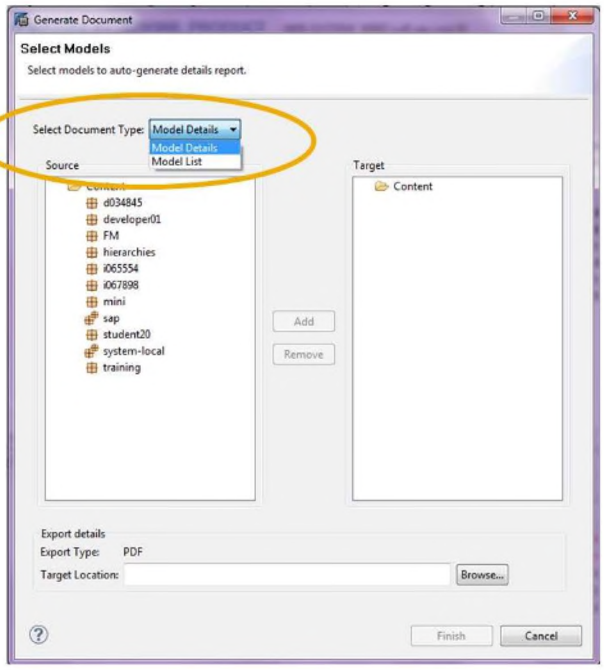


Figure 333: Auto Documentation - Select Document Type

Auto Documentation - Adding Objects to Target List

- All Information Models in the Content are available.
- You have to select one or several objects and use the "Add" button.
- Use the button "Remove" to cancel the selection.
- You can add objects from different packages in the same Auto Documentation job.

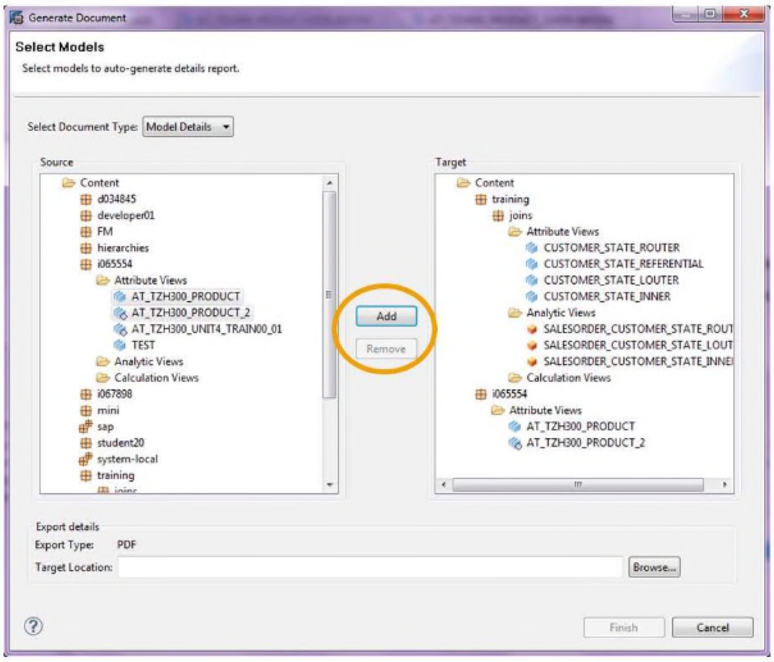


Figure 334: Auto Documentation - Adding Objects to Target List

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

Auto Documentation - Select Export Type and Save to Location

- It is not possible to change the export file type. Only .pdf type is supported.
- Choose a target location to save the generated documents.
- A dedicated folder is created in the Target Location for each System ID.

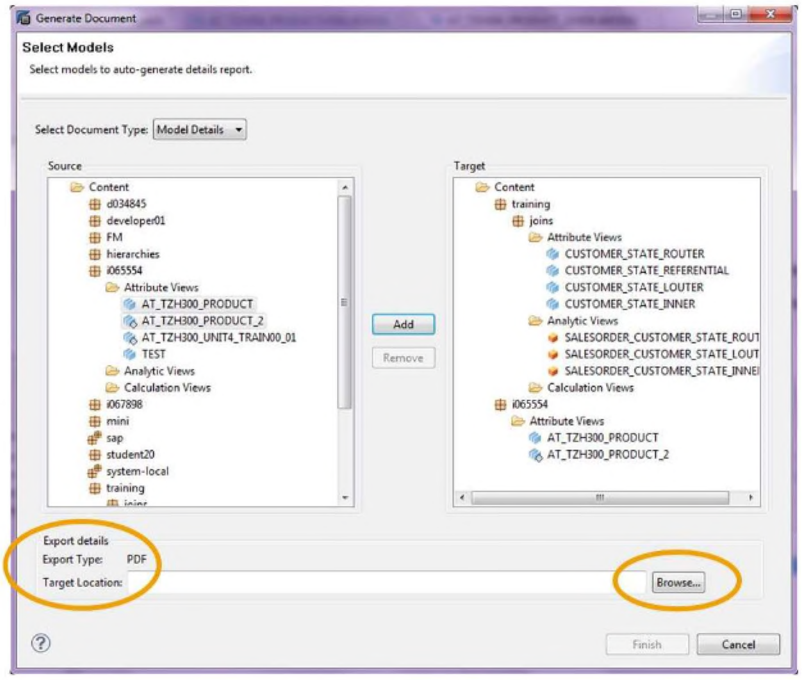


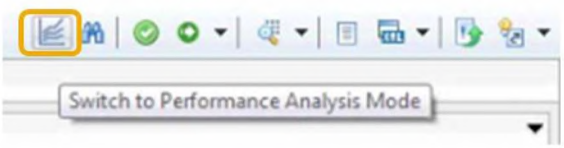
Figure 335: Auto Documentation - Select Export Type and Save to Location

Performance Analysis

The performance analysis mode helps developers to analyze their information models from a performance standpoint and to identify performance issues that might occur.

The Performance Analysis Mode can be switched on for a specific model. You can also set a preference in SAP HANA studio to open any information model with the Performance Analysis Mode switched on.

- Performance Analysis Mode provides hints and indicators inside the HANA Model Editor:
  - Join and cardinality validation
  - Table Partitioning
  - Number of Rows



- It can be switched on manually for a specific model, or automatically whenever you open a model (SAP HANA Studio Preferences).

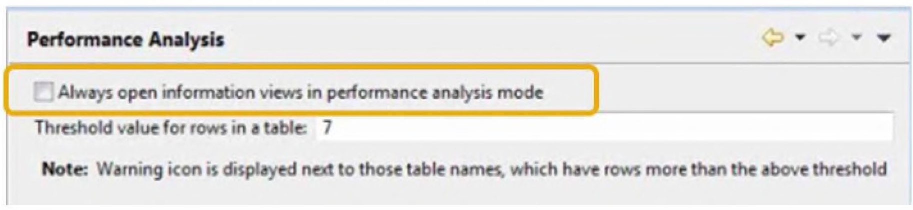


Figure 336: Performance Analysis Mode (1)

Performance Analysis Mode (2)

In the details, you can find information about the number of rows and the partition type. You can also check that the Join properties are correct, in particular by comparing the current

cardinality with the proposed one, which is based on an analysis of the content of the joined tables or sources..

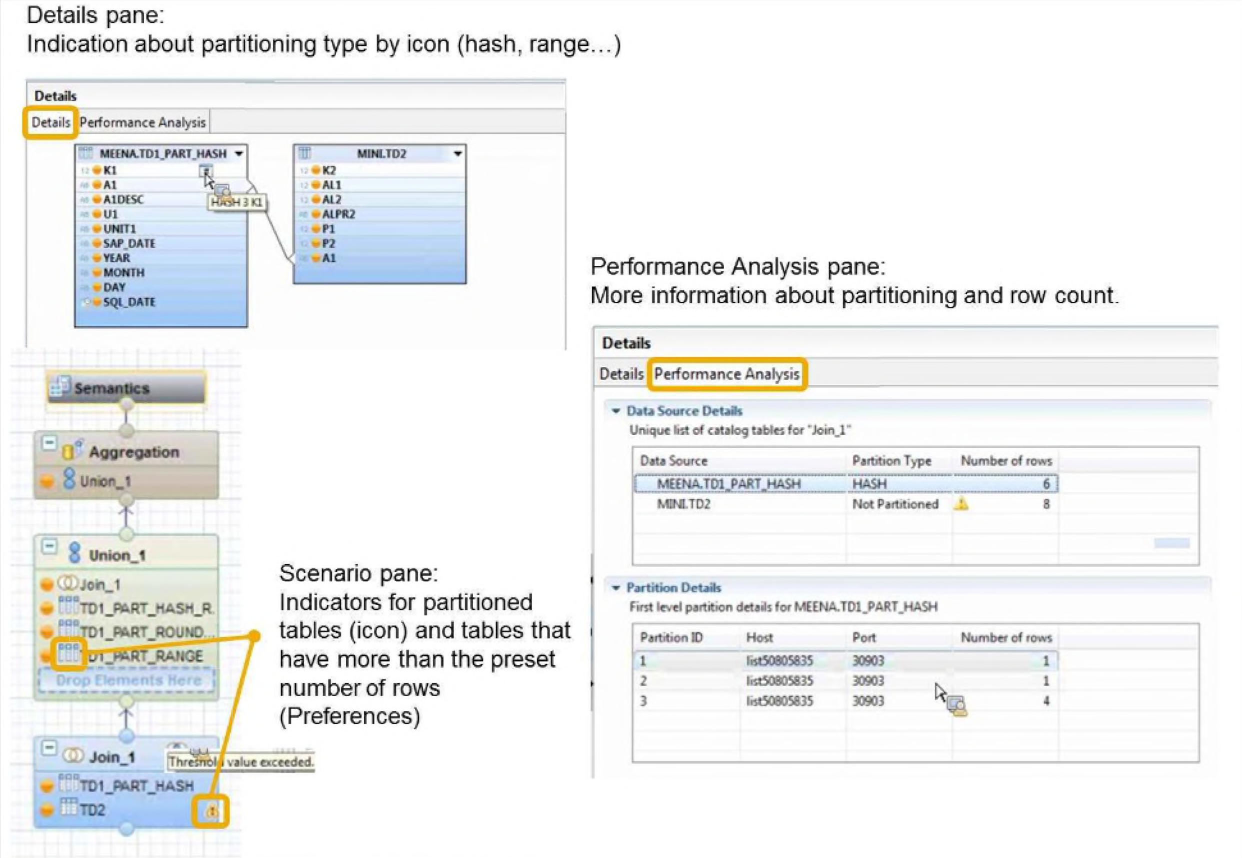


Figure 337: Performance Analysis Mode (2)

The Performance Analysis Mode provides different types of information and levels of details in the Scenario, Details, and Performance Analysis panes.

Performance-Related Validation Rules

Some Validation Rules are specifically defined in SAP HANA Studio to help you optimize model performance. They focus on the key design options that might have an impact on the performance of the views.

For example, using a calculated or aggregated column in the expression of a filter can have a negative impact on performance, and should be avoided whenever possible. The rule *Calculation in filter expression* performs the corresponding check.

Table 21: Validation Rules for Model Performance

Rule	Corresponding Best Practice for Model Design
Calculation in filter expression rule	Avoid defining filters that use calculated or aggregated columns
Calculation in joins rule	Avoid defining joins on calculated or aggregated columns
Partition types in join rule	When tables are joined, it is better if the partitions are defined on the same columns and have the same number of partitions.

Unit 6
Exercise 26

Process Information Models

Exercise Objectives

After completing this exercise, you will be able to:

- Validate an Information Model and analyze warnings and errors
- Set your own preferences in Validation Rules

Business Example

You are an SAP HANA developer and you would like to set some specific preferences in the Validation Rules for your Information Models.

Task 1: Create and Validate an Information Model

1. In the SAP HANA Studio *Preferences*, activate all *Validation Rules* for Server Version *SPS7 Onwards*.

2. Create a new Dimension Calculation View with the following details:

Property	Value
Name	**CVD_PO_BP_##**
Description	**Business Partner of a Purchase Order**
Data Category	*DIMENSION*

3. Add a Join node *Join_1* to the *Scenario* pane, and add the following tables as data sources for the node:

Table Name	Description
EPM_MODEL.SNWD_PO	Purchase Orders
EPM_MODEL.SNWD_BP	Business Partners

4. Add the following columns to the output:

Table Name	Column Name
SNWD_PO	*PO_ID*
SNWD_BP	*BP_COMPANY_NAME*
SNWD_BP	*BP_CURRENCY_CODE*

5. Join the two tables as follows with a *Referential* join and a n..1 cardinality:

Left Element	Right Element
SNWD_PO.CLIENT	*SNWD_BP.CLIENT*
SNWD_PO.PARTNER_GUID	*SNWD_BP.BP_ID*

6. Define the *Join_1* node as the data source for the *Projection* node, and all columns from the *Projection* node to the output.
7. Define the column *PO_ID* as a Key.
8. Validate your model and explain why you get an error.

Task 2: Deactivate a Validation Rule

To validate the model, you will now deactivate the rule that generated an error.

1. Disable the Validation Rule *Join → Join column Data Type Rule* in the preferences of the SAP HANA studio.
2. Validate your model again. Do you get an error again?

> Caution:
> At this stage, you have disabled the validation rule that had generated an error. However, in a real-life scenario, you must first analyze whether the reported error may generate further issues.
>
> The recommendation is to define carefully which validation rules you want to execute, based on your specific business requirements, and stick to the defined set of rules.

> Note:
> An intermediate "solution" object is available at this stage:
>
> *STUDENT00 → ha300 → CVD_PO_BP_00_STAGE1*

Task 3: Modify the Set of Validation Rules and Correct the Information Model

You will now modify the set of Validation Rules and also modify the inconsistent join from the Information Model.

1. Observe the remaining warnings that you get after the successful validation.
2. Add the *CLIENT* column from table *SNWD_PO* to the output of the view (as the first column) and define it as a key column.

> Hint:
> To avoid adding the column to both the *Join_1* and *Projection* nodes, you can add it to the *Join_1* output only, and then use the command *Propagate to Semantics*.

3. Modify the joined columns so that the *SNWD_PO.PARTNER_GUID* column is connected to the *SNWD_BP.NODE_KEY* column.
4. Make the following changes to the Validation Rules settings.

- Activate the validation rule *Join → Join column Data Type Rule* in the preferences of the SAP HANA studio.
- De-activate the validation rule *Content → Description Check*
- De-activate the validation rule *Content → Naming Convention Check*

5. Validate your model again and observe the validation log. What is similar or different compared with the previous validation?

Unit 6
Solution 26

Process Information Models

Exercise Objectives

After completing this exercise, you will be able to:

- Validate an Information Model and analyze warnings and errors
- Set your own preferences in Validation Rules

Business Example

You are an SAP HANA developer and you would like to set some specific preferences in the Validation Rules for your Information Models.

Task 1: Create and Validate an Information Model

1. In the SAP HANA Studio *Preferences*, activate all *Validation Rules* for Server Version *SPS7 Onwards*.
   a) In SAP HANA Studio, choose *Window → Preferences*.
   b) Navigate to the node *SAP HANA → Modeler → Validation Rules* and in the *Server Version* drop-down list, select *SP7 onwards*.
   c) In the lower part of the screen, choose *Check all*.
   d) Choose *OK* to close the screen.
2. Create a new Dimension Calculation View with the following details:

Property	Value
Name	**CVD_PO_BP_##**
Description	**Business Partner of a Purchase Order**
Data Category	*DIMENSION*

   a) Right-click your package and choose *New → Calculation View*.
   b) Enter name and description as shown in the table.
   c) Choose *Finish*.
3. Add a Join node *Join_1* to the *Scenario* pane, and add the following tables as data sources for the node:

Table Name	Description
EPM_MODEL.SNWD_PO	Purchase Orders

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

Table Name	Description
EPM_MODEL.SNWD_BP	Business Partners

a) Choose the green plus sign on the right of the *Join_1* node and add the two tables.

4. Add the following columns to the output:

Table Name	Column Name
SNWD_PO	*PO_ID*
SNWD_BP	*BP_COMPANY_NAME*
SNWD_BP	*BP_CURRENCY_CODE*

a) Right-click each of the fields indicated in the table, and select *Add to Output* from the context menu.

5. Join the two tables as follows with a *Referential* join and a n..1 cardinality:

Left Element	Right Element
SNWD_PO.CLIENT	*SNWD_BP.CLIENT*
SNWD_PO.PARTNER_GUID	*SNWD_BP.BP_ID*

a) In the *Details* pane of the *Join_1* node, drag the columns from the left table to the join column of the right table, as specified in the table.

b) Double-click any of the connectors between the two tables

c) Set the Join Type to *Referential* and the Cardinality to *n..1* and choose *OK*.

6. Define the *Join_1* node as the data source for the *Projection* node, and all columns from the *Projection* node to the output.

a) In the *Scenario* pane, draw a connector from the top of the *Join_1* node to the *Projection* node.

b) Select the *Projection* node and in the *Details* pane, right-click the *Join_1* table and choose *Add All to Output*.

7. Define the column *PO_ID* as a Key.

a) Choose the *Semantics* node.

b) Select the *Key* checkbox for the *PO_ID* column.

8. Validate your model and explain why you get an error.

a) Choose *Save and Validate*.
You get an error because you joined fields of different types in the *Join_1* node and the Validation Rule *Join column Data Type Rule* is active.

Task 2: Deactivate a Validation Rule

To validate the model, you will now deactivate the rule that generated an error.

1. Disable the Validation Rule *Join → Join column Data Type Rule* in the preferences of the SAP HANA studio.

a) Reopen the *Validation Rules* preferences in the same way as the first step of this exercise.

b) Deselect the Validation Rule *Join → Join column Data Type Rule*.

c) Choose *OK*.

2. Validate your model again. Do you get an error again?

a) Choose *Save and Validate*.
The model is activated successfully because the Validation Rule is now disabled. In the job log, the status is *Completed with Warnings*, because any time a validation rule is disabled, a warning is generated to inform the user that only a sub-set of the validation rules has been applied.

Caution:
At this stage, you have disabled the validation rule that had generated an error. However, in a real-life scenario, you must first analyze whether the reported error may generate further issues.

The recommendation is to define carefully which validation rules you want to execute, based on your specific business requirements, and stick to the defined set of rules.

Note:
An intermediate "solution" object is available at this stage:

STUDENT00 → ha300 → CVD_PO_BP_00_STAGE1

Task 3: Modify the Set of Validation Rules and Correct the Information Model
You will now modify the set of Validation Rules and also modify the inconsistent join from the Information Model.

1. Observe the remaining warnings that you get after the successful validation.

a) If needed, from the *Job Log* view, open the *Model validation job details* again.
The job details show the following warnings:

- *Description Check* (Attribute Labels).
- *Naming Convention Check* (No specific naming convention is defined in your SAP HANA System).
- *Client Field Join Check* (the CLIENT column is not part of the output).
- *Skipped: Join Column Data Type Rule* (this Validation rule has been ignored).

2. Add the *CLIENT* column from table *SNWD_PO* to the output of the view (as the first column) and define it as a key column.

> Hint:
> To avoid adding the column to both the *Join_1* and *Projection* nodes, you can add it to the *Join_1* output only, and then use the command *Propagate to Semantics*.

a) In the *Join_1* node, right-click the *CLIENT* column from source table SNWD_PO and choose *Add to Output*.

b) In the *Output* pane, move the *CLIENT* column to the top of the list.

c) Right-click the *CLIENT* column and choose *Propagate to Semantics*. Choose *OK* to validate the message.

d) In the *Semantics* node, define the *CLIENT* column as a key.

e) If needed, move the *CLIENT* column to the top of the list.

3. Modify the joined columns so that the *SNWD_PO.PARTNER_GUID* column is connected to the *SNWD_BP.NODE_KEY* column.

   a) In the *Join_1* node, drag and drop the corresponding connector to the *NODE_KEY* column of the right table.

   b) Alternatively, you can delete and recreate the connection between the columns *SNWD_PO.PARTNER_GUID* and *SNWD_BP.NODE_KEY*.

4. Make the following changes to the Validation Rules settings.

   - Activate the validation rule *Join → Join column Data Type Rule* in the preferences of the SAP HANA studio.
   - De-activate the validation rule *Content → Description Check*
   - De-activate the validation rule *Content → Naming Convention Check*

   a) Open the *Validation Rules* preferences as in the first step of this exercise.

   b) Select or Deselect the validation rules as specified.

   c) Choose *OK*.

5. Validate your model again and observe the validation log. What is similar or different compared with the previous validation?

   a) Choose *Save and Validate*.
   Like after the previous validation, the model is activated successfully and its status is *Completed with Warnings*.

   However, this time, the Client-side validation is free of any warning or error because the model was modified. The only warning you get corresponds to the list of rules that were skipped because they are deactivated.

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

LESSON SUMMARY

You should now be able to:

- Process and analyze models
- Compare different versions of an Information Model
- Check model references
- Generate auto documentation
- Execute a Performance Analysis of an Information Model

Unit 6
Lesson 2

Managing Modeling Content

LESSON OVERVIEW

This lesson explains how you can transport modeling content between two different SAP HANA systems.

You will also learn how to move and rename modeling content within one SAP HANA system.

Business Example

You are working at a client site and need to move Information Models from the development to the productive system landscape.

You need to understand how to perform import and export of models as well as how to manage schema mappings.

You also need to move or copy information models and still keep the overall repository consistent.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Manage modeling content
- Define package-specific authoring schema
- Transport information models between two systems
- Move and copy information models within a system
- Take over an information model edited by another user
- Translate metadata texts

Schema Mapping

A schema is used in SAP HANA to group database objects, such as Tables, Views, or Procedures.

In the information models you create, the source objects that you use (in particular, tables) are chosen from a specific schema, either explicitly (for example, when you add a table to the data foundation of an Attribute View), either implicitly, when you assign a default schema to a script-based Calculation View or a Procedure.

If you need to transport an Information Model from a source system to a target system, you need to take care of the schemas where the tables used by the Information Model are stored.

The Information Model could point to a table that is not located in the same schema in the source and in the target system. If this is the case, before importing the Information Model in the target system, you need to define a *Schema Mapping*, where you map the source schema to the target schema. In the context of Schema Mapping, the source and target schemas are referred to as follows:

Key Vocabulary of Schema Mapping

- Authoring Schema

  The schema name in the source system.

- Physical Schema

  The schema name in the target system.

> Note:
> The Schema mapping defined in the system also applies to the *Default Schema* property of Analytic and Calculation Views.
>
> This mapping is particularly useful in script-based calculation views because you can name the tables without mentioning the source schema in the SQL syntax (tables without explicit schema name will be read from the Default Schema assigned to the view). In case you transport the script-based Calculation View to another system where tables are located in a different schema, schema mapping applies, keeping your Calculation View valid in the target system.

From SAP HANA SPS10 onwards, the definition of a schema mapping supports External databases in a Multi Database Container SAP HANA System.

Creating a Schema

Before you can use schema mapping, you first need to define the source and target schemas. There are various ways to create a schema as follows:

- You can create it with the `CREATE SCHEMA` SQL statement in the SAP HANA studio.
- Schemas are usually automatically created when you install an application on SAP HANA. For example, when you implement suite on HANA or S/4HANA, the SAP HANA schema is automatically created during the installation phase.
- There is also a schema created and named for each SAP HANA user when the user ID is first created.

An administrator will typically maintain the Schema Mapping in the target system before importing the packages that contain the Information Models.

- **You create schemas to group the tables. For import you need to create the schema where all the tables are imported.**
  - Schemas are created with a SQL Script statement.
  - SQL Syntax: `CREATE SCHEMA schema_name [OWNED BY name]`
  - Parameters: *`OWNED BY`* Specifies the name of the schema owner.
  - Description: The `CREATE SCHEMA` statement creates a schema in the current database.

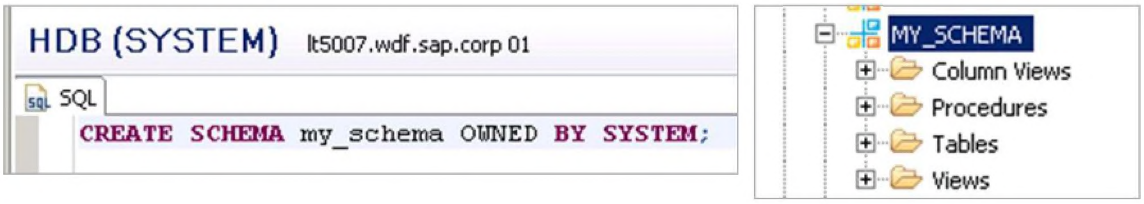


Figure 338: Creating a Schema

Example:

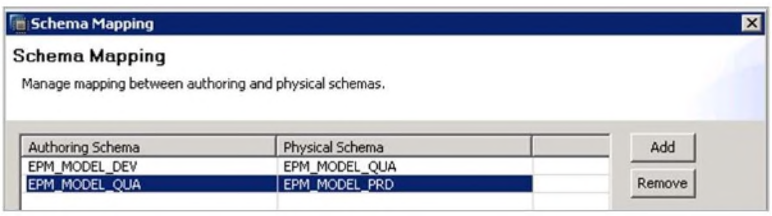
```
CREATE SCHEMA my_schema OWNED BY system
```

Defining Schema Mapping

When the *Schema* has been created in the target system, you can, as a first step, import the relevant tables in it. After that, and before importing the Information Models, you need to define the Schema Mapping.

To define a schema mapping, in the *Quick View* pane of the *SAP HANA Modeler* perspective, choose *Schema Mapping*.

- You use this procedure to map the *Authoring Schemas* to the *Physical Schemas* in the target system, when transporting Information Models from a source system to a target system, or when importing SAP-shipped content.
- All *Schema Mappings* are maintained in the same window.



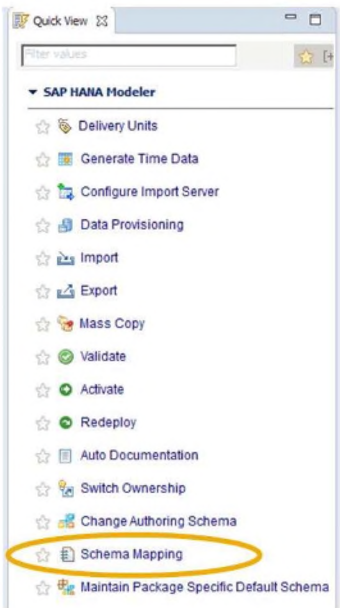


Figure 339: Defining Schema Mapping

Transporting Tables to a Different Schema

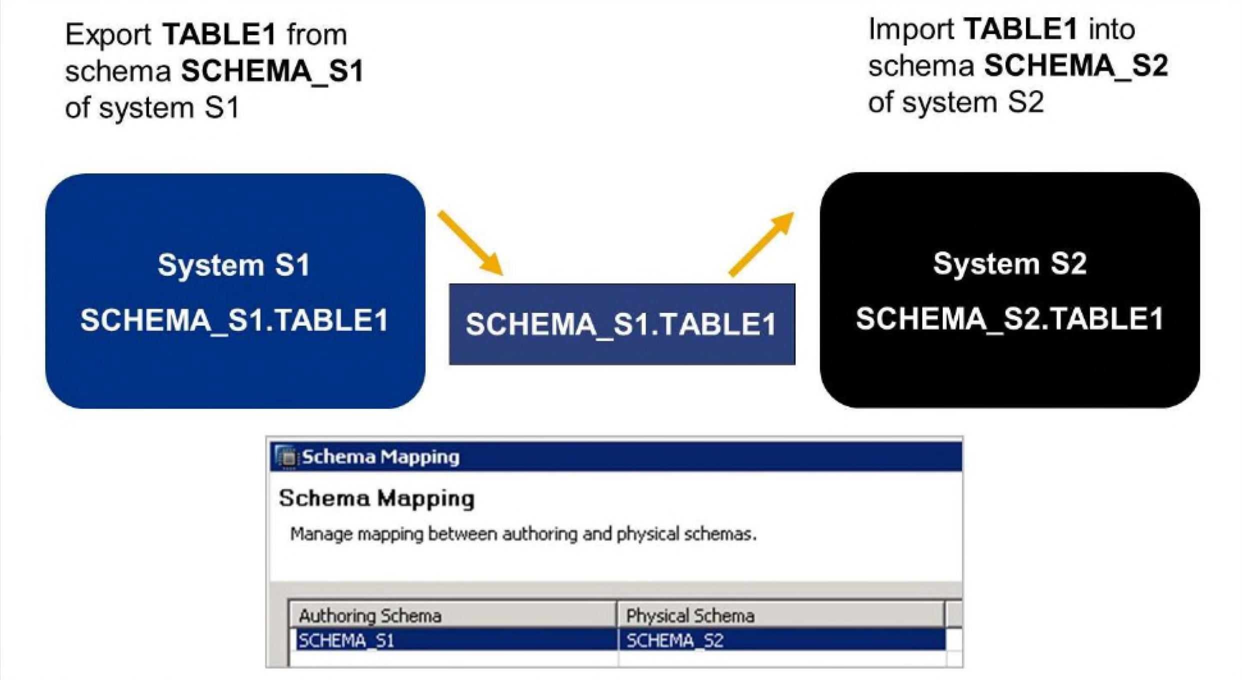


Figure 340: Transporting Tables to a Different Schema

The figure, Transporting Tables to a Different Schema, describes a scenario where a table *TABLE1* has been transported from schema *SCHEMA_S1* in a source system to schema *SCHEMA_S2* in a target system. Now, you want to transport an Information Model referring to the table *TABLE1* in physical schema *SCHEMA_S1* from the source system to the target system.

- If you export the Information Model from the source system and import it into the target system, without defining the relevant *Schema Mapping*, then the imported Information Model will not work because it will still try to read its tables from Schema *SCHEMA_S1*, which does not exist in the target system.
- To prevent this issue, you must maintain the Schema Mapping, to map the Authoring Schema *SCHEMA_S1* to the Physical Schema *SCHEMA_S2*.
- The Information Model can be imported successfully now and, after the import, it will correctly point to schema *SCHEMA_S2*.

Analytic View with a Table in Schema HA300

For example, an Information View in the source system exists with a *SALES* table in schema *HA300*.

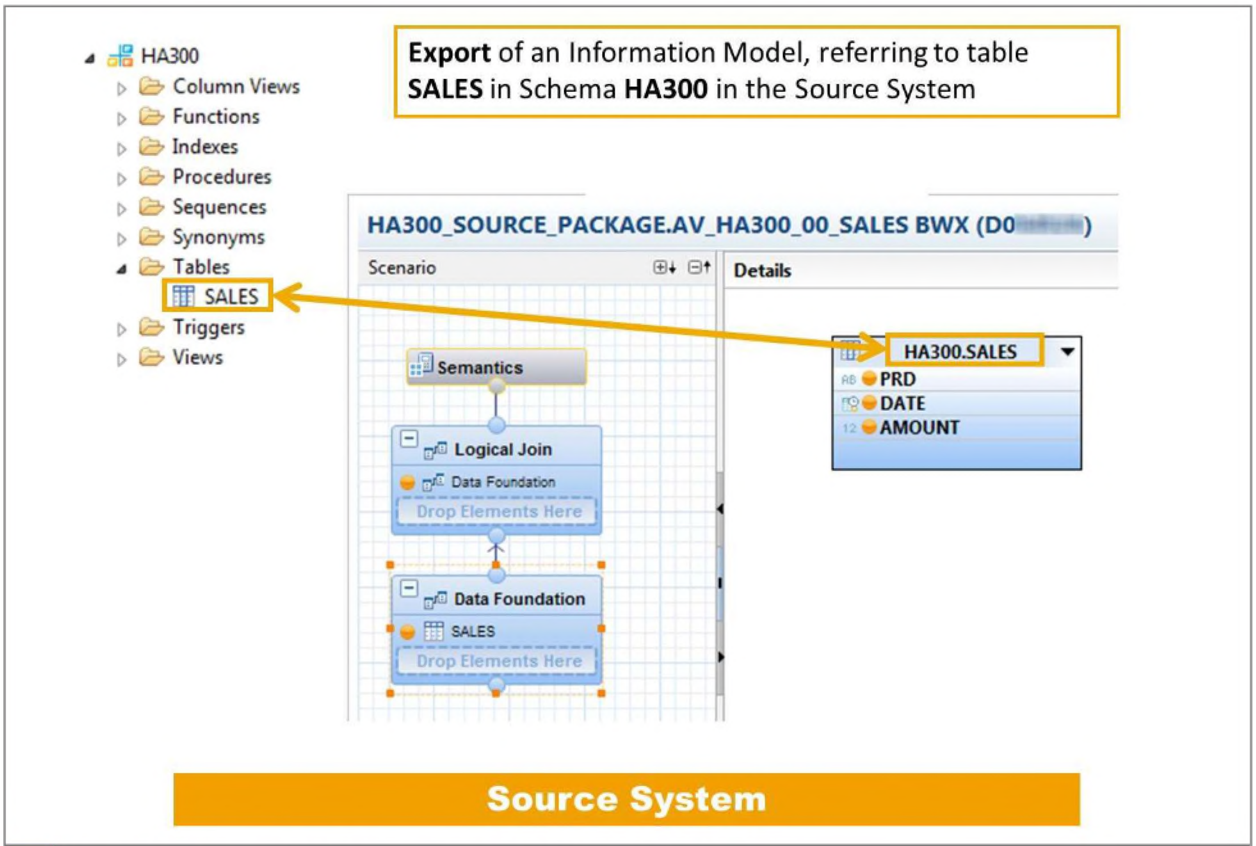


Figure 341: Information View with a Table in Schema HA300

Schema Mapping and Imported Analytic View in the Target System

You want to import this Analytic View in a target system, where the table *SALES* is stored in schema *STUDENT04*. Before importing the model, you need to map the Authoring Schema *HA300* to the Physical Schema *STUDENT04* in the target system.

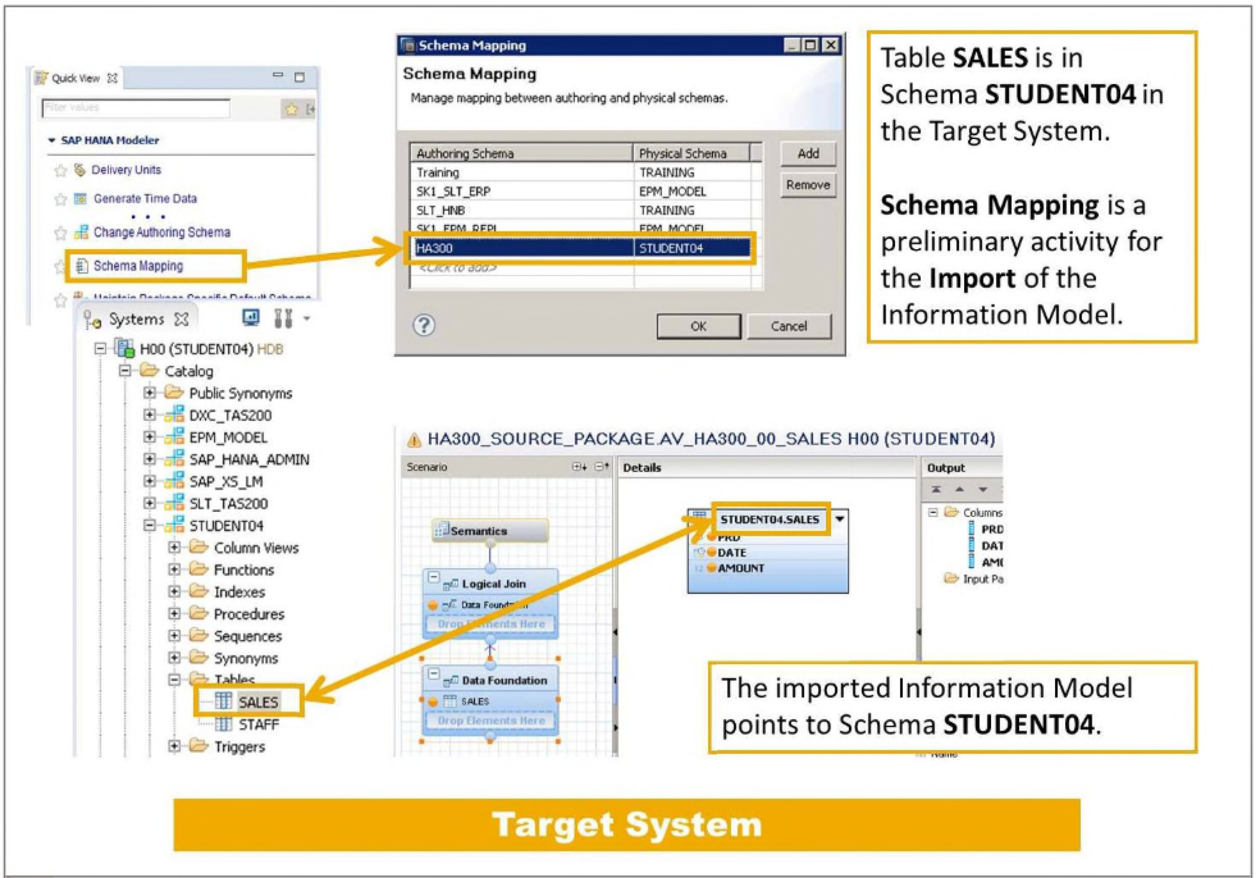
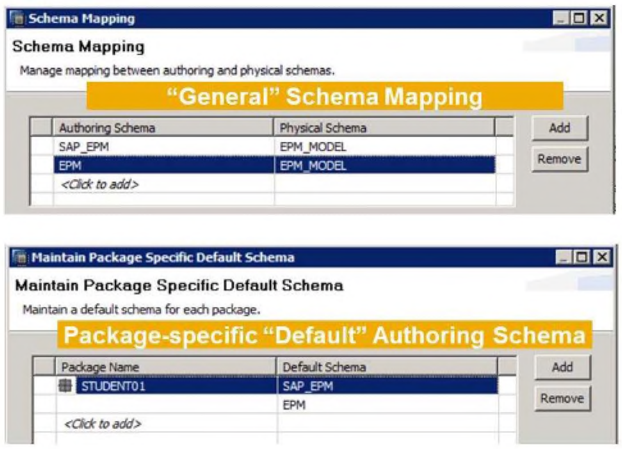
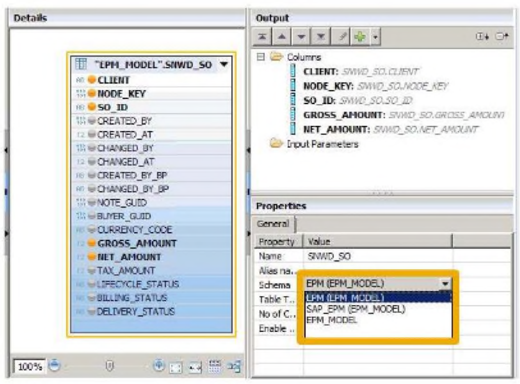


Figure 342: Schema Mapping and Imported Analytic View in the Target System

Schema Mapping - Defining Package-Specific Authoring Schema

- It is possible to define a package-specific default authoring schema.
- This setting is useful when two different authoring schemas are mapped to the same physical schema:
  When adding a table from the physical schema to an information view, SAP HANA identifies automatically the relevant authoring schema.





- Additionally, the *Schema* property of a catalog object in the model editor shows the physical schema (and the mapping to the authoring schema, if any) and can be modified.

Figure 343: Schema Mapping - Defining Package-Specific Authoring Schema

In some scenarios, several Authoring Schemas might be mapped to the same Physical Schema. In this case, it is possible to define which Authoring Schema should be considered as the default one at the package level.

In the example, the physical schema *EPM_MODEL* corresponds to the authoring schemas *SAP_EPM* and *EPM*.

When you create an information view that uses any table from physical schema *EPM_MODEL*, SAP HANA studio cannot determine which authoring schema the information model should actually reference.

A package-specific default authoring schema has been defined. As a consequence, if you create an information model in package *STUDENT01* (or any of its sub-packages), any reference to the physical schema *EPM_MODEL* will be interpreted in the design of the model as a reference to the authoring schema *SAP_EPM*.

> Note:
> As shown in the example, in the *Package Specific Default Schema* window, it is also possible to create an entry with an authoring schema but no package name. This authoring schema will be considered as the default for any information model that is not created within the package *STUDENT01*.

Exporting and Importing Information Models

Information Models are stored in *Packages* and to export and import *Packages*, they must be included in a *Delivery Unit*.

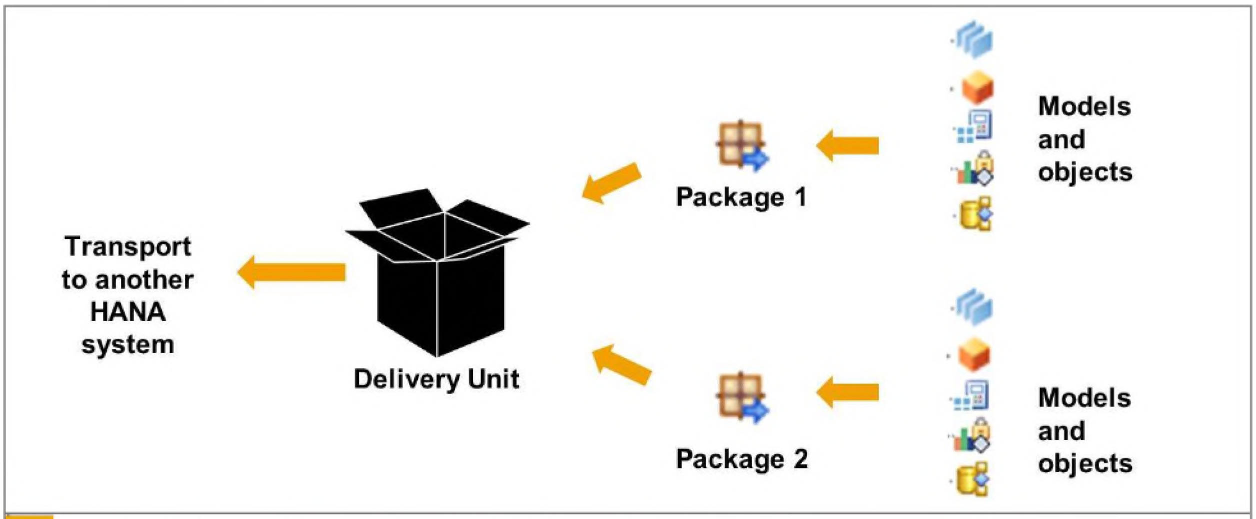


Figure 344: Delivery Units, Packages and Information Models

> Note:
> Remember, only the modeling content from Packages can be included in the import and export by Delivery Unit. Other types of content, such as tables and users can be imported and exported using other techniques.

Export and Import

- You can import models from your local system or from a server.

  Procedure:

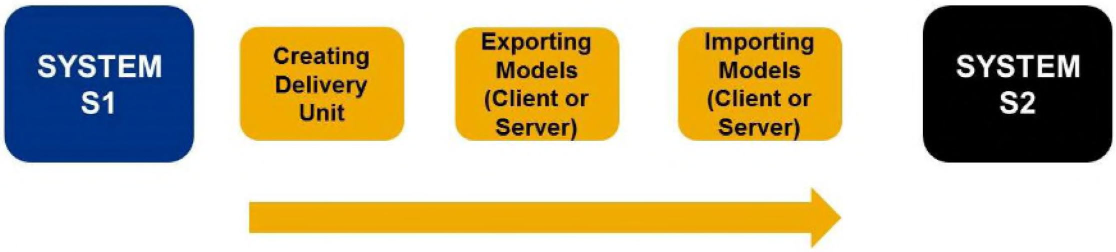


Figure 345: Export and Import

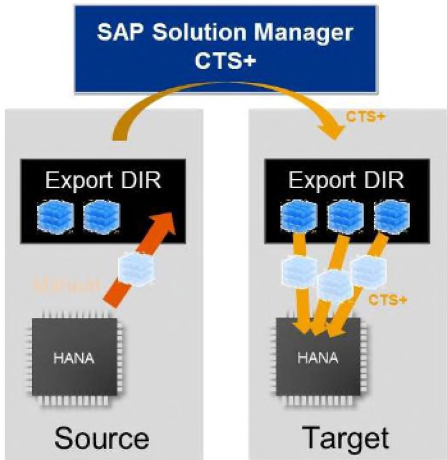
HANA Content Transport Capabilities

In the context of Data Marts:

- 2-step integration into CTS+ ("loose coupling"):
  - Manual preparation (server-side export)
  - Automated transport and deployment in target system via CTS+
- Potential for ABAP-based new applications (HPAs)
  - TLOGO-based transport
  - Encapsulating SAP HANA content in ABAP objects allows to transport SAP HANA content with application code though standard CTS mechanisms

In the context of SAP NW 7.3 BW, powered by SAP HANA

- Leverage existing transport functionality



CTS = Change and Transport System

HPA = HANA-Powered Applications

Figure 346: HANA Content Transport Capabilities

Creating a Delivery Unit

- You use a Delivery Unit to create a group of transportable objects for content delivery and to export information models from source system to target server.
- From the Quick View pane, click *Delivery Units*

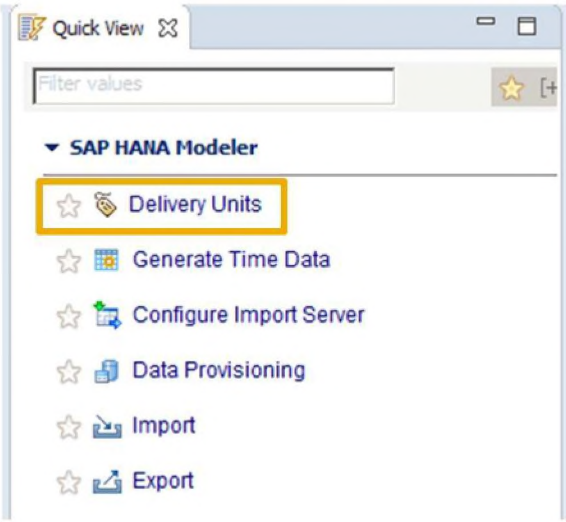
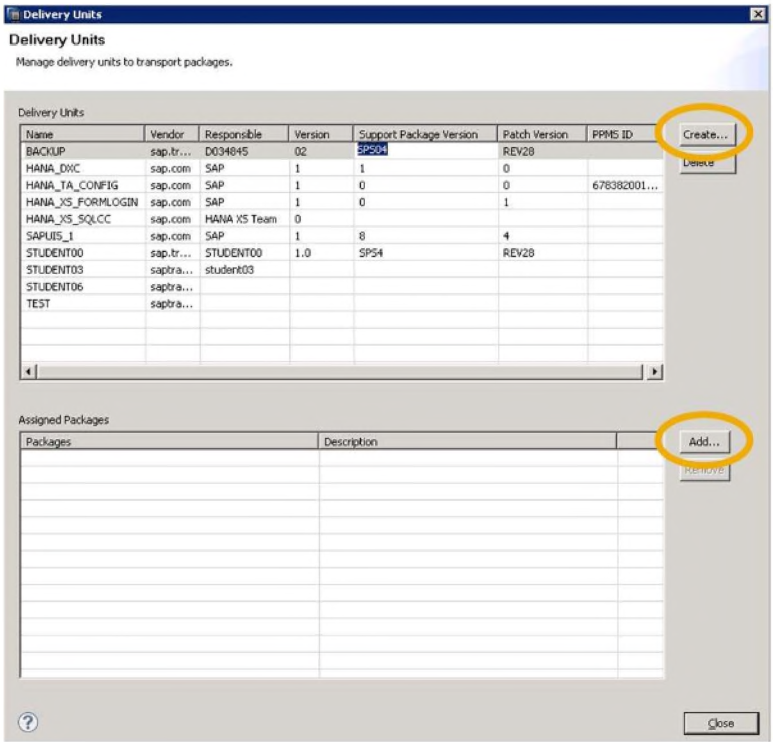


Figure 347: Creating a Delivery Unit

Assigning Packages to a Delivery Unit



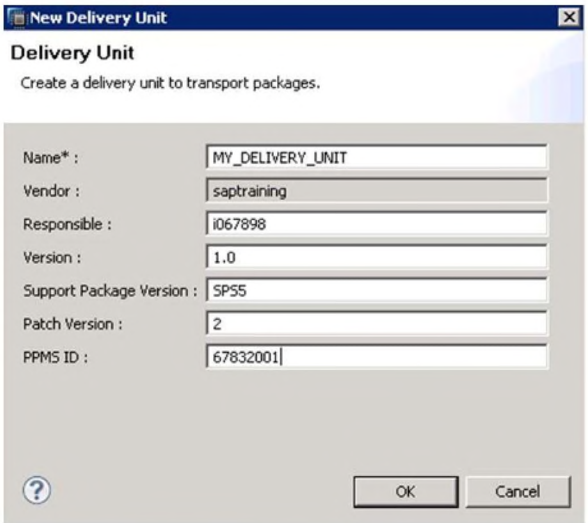
- From the Delivery Units dialog window, click *Create*.
- You will also use this window to assign Packages to the Delivery Units.

Figure 348: Assigning Packages to a Delivery Unit

The same window is used to manage delivery units (create, modify, or delete) and to assign Packages to Delivery Units.

Note:
It is also possible to assign a Package to an existing Delivery Unit when you create the Package (in the *New Package* window).

Delivery Unit Properties



- Enter the delivery unit name.
- Enter the responsible user.
- In the Version field, enter the delivery unit version.
- Enter the support package version of the delivery unit.
- Enter the patch version of the delivery unit.
- Enter an optional PPMS ID

Figure 349: Delivery Unit Properties

Note:
The PPMS ID is one of the attributes of a *Delivery Unit*. The Product and Production Management System (PPMS) is the central knowledge repository for (technical) information regarding SAP software products and their software components.

Exporting a Delivery Unit

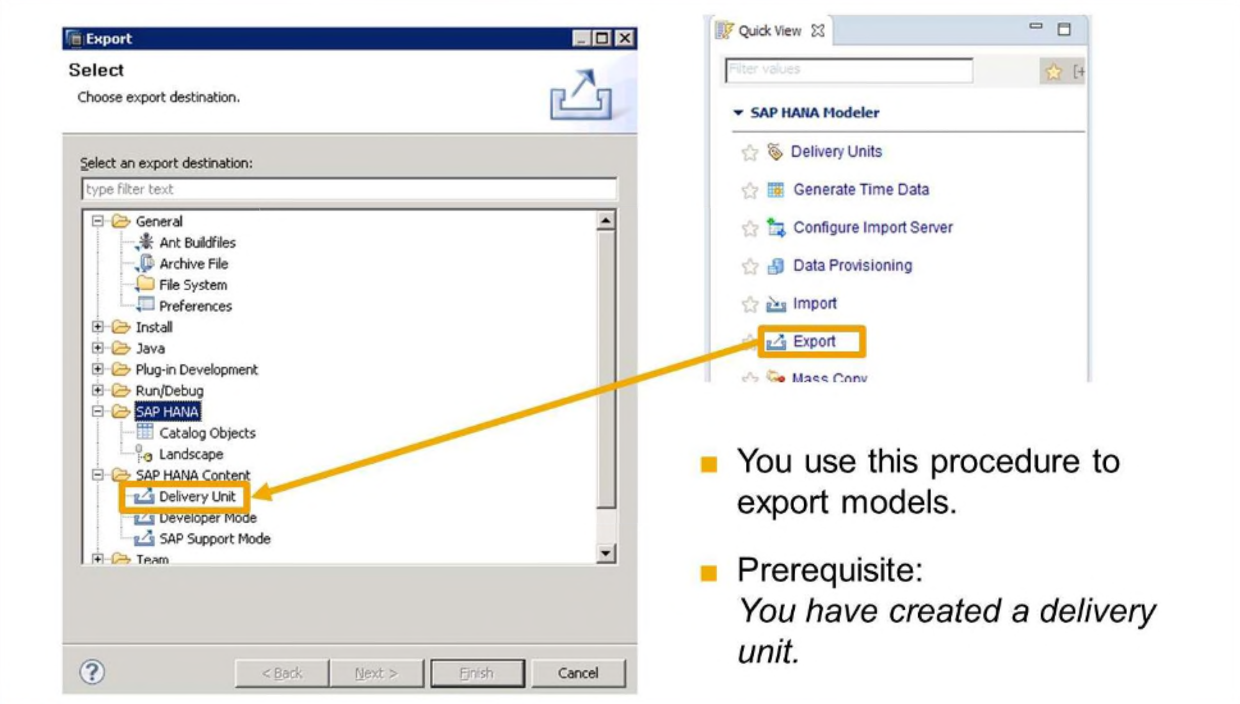


Figure 350: Exporting a Delivery Unit

Exporting objects using *Delivery Units* includes the following:

- Function to export all packages that make up a *Delivery Unit* and the relevant objects contained therein, to the client or to the SAP HANA server file system.
- Exporting objects using *Developer Mode*:

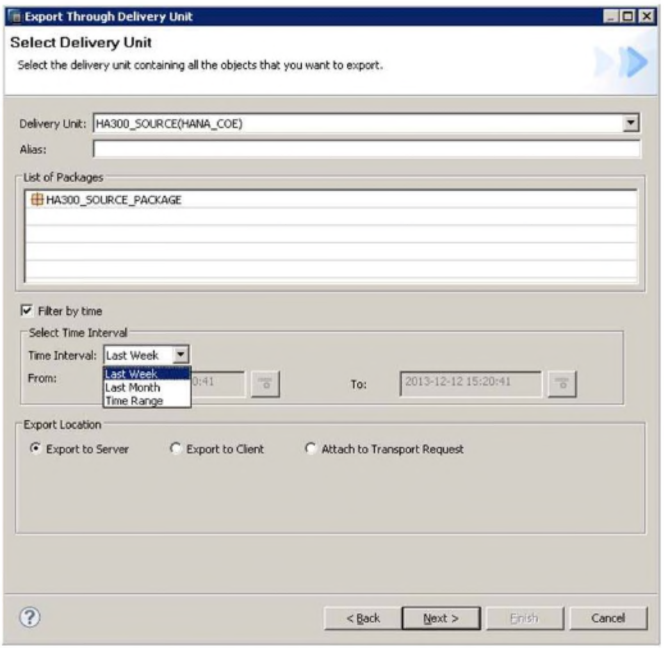
  Function to export individual objects to a directory on your client computer. This mode of export should only be used in exceptional cases because it does not cover all aspects of an object, for example, translatable texts are not copied.

Importing objects using *Delivery Unit*:

- Function to import objects (grouped into a *Delivery Unit*) from the server or client location available in the form of .TGZ file.
- Importing objects using *Developer Mode*:

  Function to import objects from a client location to an SAP HANA modeling environment.

Checking List of Exported Packages and Time Filter



- Select the *Delivery Unit*.
- Then choose whether to export to the server or the client and click on "Next" button.
- *Note that you can also filter by Selecting a Time Interval.*

Figure 351: Checking List of Exported Packages and Time Filter

A careful approach is needed with the *Filter By Time* option to avoid serious object consistency problems. It is a general best practice to periodically schedule full exports, and have only a few exports using *Filter By Time* in between. When using *Filter By Time*, it is recommended to use a *From* date that corresponds to the date of the last full export. To ensure that no object changes are missed, and therefore that a consistent export is performed, it's best to use a date and time that is slightly before the date and time of the last full export.

Importing a Delivery Unit

- You use this procedure to import Information Models from the server or the client.

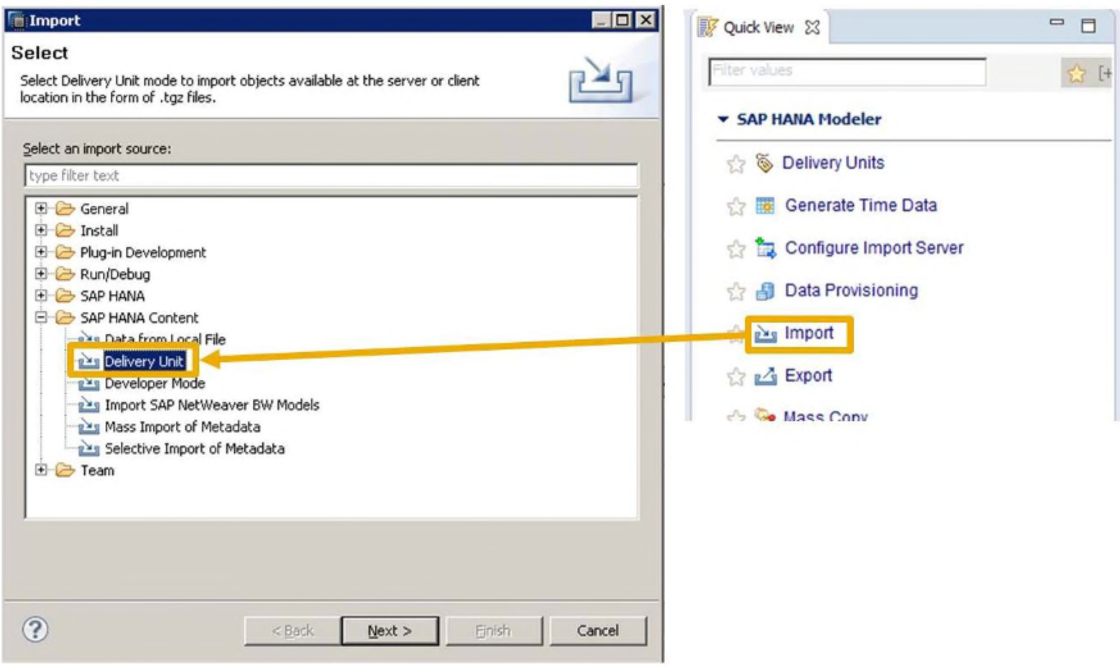


Figure 352: Importing a Delivery Unit

Checking List of Imported Models and Import Actions

- Select the file repository on the client or on the server where models have been exported.
- Then select the models you want to import.
- Set the checkboxes “Overwrite inactive versions” and “Activate objects” and then click the “Finish” button

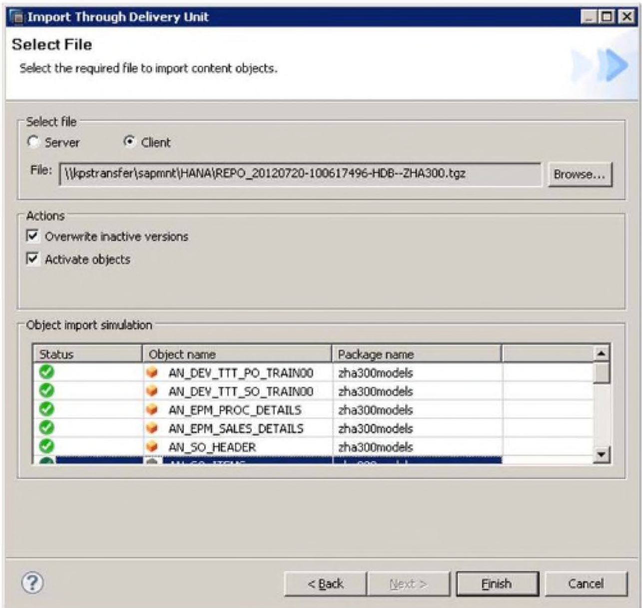
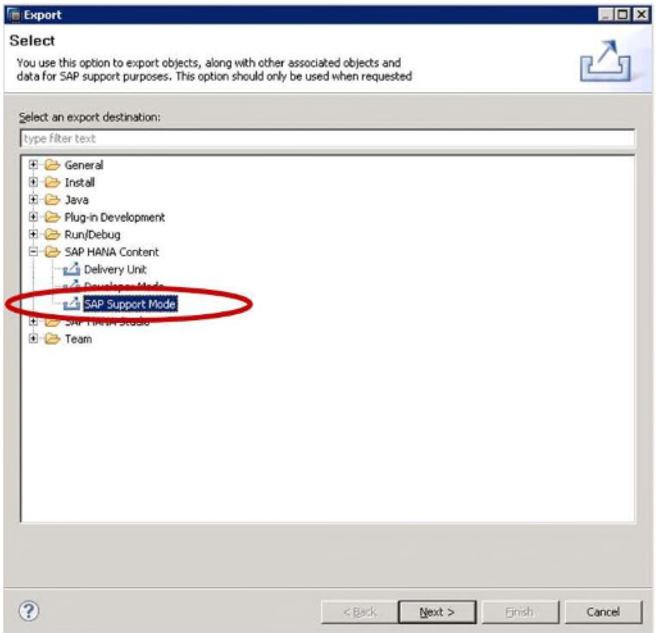


Figure 353: Checking List of Imported Models and Import Actions

- To overwrite inactive objects in the target system, select the option *Overwrite inactive versions*.
- To activate objects in the target system, select the option *Activate Objects*.
- The status of import can be viewed in the Job log.

Hint:
If you want to activate the imported objects, you need to grant *SELECT* privileges to user *_SYS_REPO*.

Import and Export – SAP Support Mode



- In order to ease communication when working together with SAP to gain support for Information Models, the export tool provides a method to export Information Objects to the server in a mode named "SAP Support Mode".
- Only active objects can be exported in this mode.
- These will be exported to the server and the file(s) can then be sent to SAP support for troubleshooting purposes.

Figure 354: Import and Export – SAP Support Mode

The status of export can be viewed in the Job log.

Copying Information Models

The options to make copies of information models are as follows:

- Perform a simple copy

  You can either use the Ctrl+C/Ctrl+V shortcut, or create a new information view and, in the *New Information View* screen, select the *Copy From* checkbox and select the information view you want to copy.

  In this case, the new object (the copy) is not referenced by any model (because it is a brand new object) and, if the source object had references to other information models, these references remain the same in the new object (the copy).

- Perform a Mass Copy

  Another option, more suited in some cases, is to copy several information models at once, and replicate the object dependencies that existed between the source objects, so that the copies have the same dependencies between one another.

The *Mass Copy* is a functionality that allows you to copy several information models at once, while keeping the references between all the copied objects consistent.

For example, if you use *Mass Copy* to copy a graphical view CV2 and an underlying graphical view CV1 from package P1 to package P2, the new (copied) view P2.CV2 will reference the new (copied) view P2.CV1.

> Note:
> On the contrary, a “simple” copy of view P1.CV2 to package P2 would create a View *P2*.CV2 that would still reference the view *P1*.CV1

Copying Information Objects

With a *Mass Copy*, you can copy the standard content shipped by SAP to your local package for your modeling and reporting use cases, for example from **`sap.ecc.fin`** to **`customer.ecc.fin`**.

> Caution:
> Copy the SAP delivered content to your own packages, to avoid overwriting changes during subsequent imports.

The *Mass Copy* functionality can also be used for your own modeling purposes (for example, to create sandbox information models without impacting the original objects).

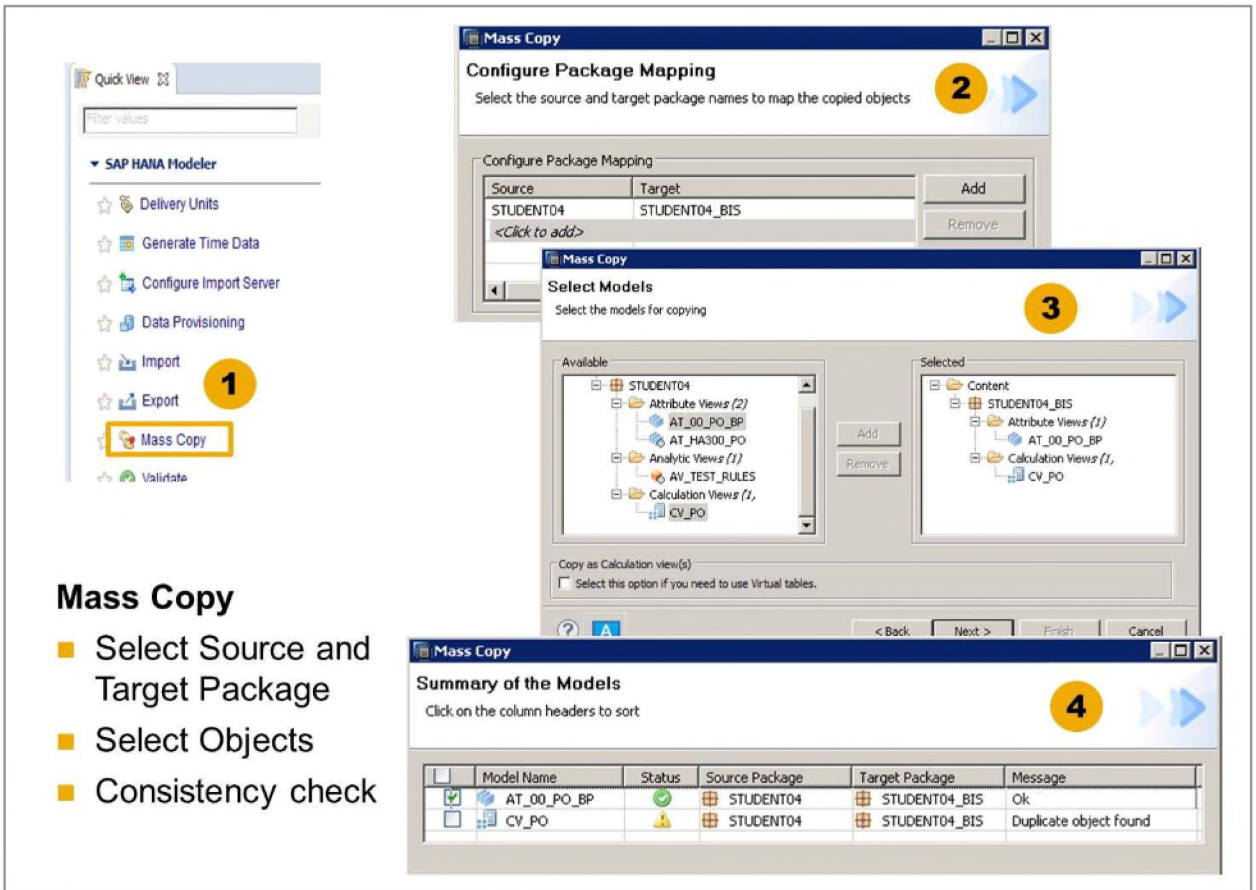


Figure 355: Copying Information Objects

Copying Content Delivered by SAP

- You use *Mass Copy* to copy standard content shipped by SAP or an SAP partner to your local repository).
- **Procedure:**

  1. From the *Quick View* pane, click *Mass Copy*.
  2. Maintain the package mappings.
  3. Choose the objects you want to copy.
  4. If needed, select the *Copy as Calculation View(s)* checkbox.
  5. Choose Finish.
- The status of content copy can be viewed in Job log.

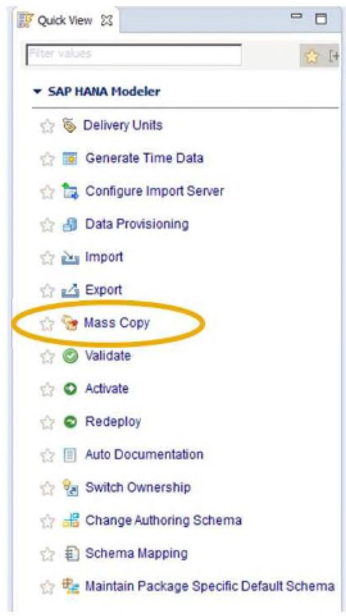


Figure 356: Copying Content Delivered by SAP

> Note:
> During the Mass Copy, if an object already exists in the target package, it is by default excluded from the copy.
>
> If you want to overwrite the existing object in the target package, you must select the *Copy* checkbox in the last screen of the *Mass Copy* dialog. The warning message *Duplicate Object Found* is then replaced by *Will overwrite*.

To copy the content, you must have the following privileges:

- *REPO.READ* on the source package
- *REPO.MAINTAIN_NATIVE_PACKAGES* on the Root Package
- *REPO.EDIT_NATIVE_OBJECTS* on the Root Package

Renaming an Information Model

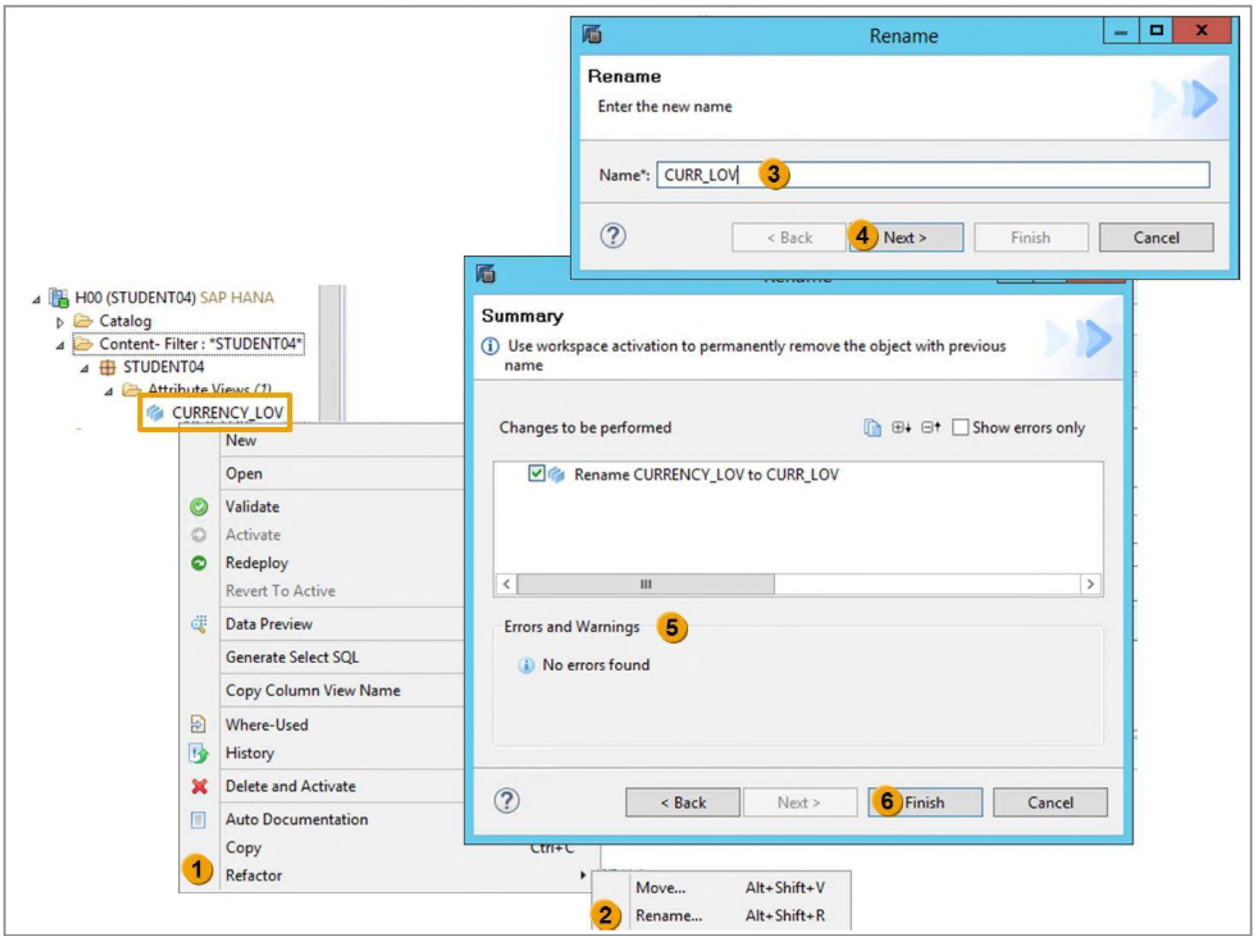


Figure 357: Renaming Information Views

It is possible to rename an information model, and adjust the references to this model within the other Information Models.

> Note:
> You can only rename one object at a time.

The functionality checks the dependencies between models, and gives the list of changes that it will perform on related Information models.

After renaming the object with the *Rename* utility, you must activate the new object from the *System* view and make sure you also activate the deletion of the previous objects.

Renaming the Columns of an Information Model

From SPS10 onwards, it is possible to rename one or several columns of an Information view, and propagate the changes to dependent views, that is, to make sure that all the graphical models that consume this information view will be adjusted accordingly.

To do that, in the Semantics of the view, use the *Rename and Adjust References* feature. A list displays all the dependent models, and you can choose the ones in which the reference to the renamed column(s) must be adjusted (all the dependent models are selected by default).

> Caution:
> The *Rename and Adjust References* functionality never impacts the SQL-based dependent objects, such as Tables Functions, Procedures, and Scripted Calculation Views. You must manually adjust the column names in the SQL code of dependent objects according to the new column names of the modified graphical view(s).

Moving Information Models

The *Refactoring* functionality allows you to move your content objects from one package to another, without changing their behavior: as you move the objects, their references are automatically adjusted.

The approaches to re-factor information models, depending on your requirements, are as follows:

- You can select one or several information models within the same package. All the information objects will be moved to another existing package.

  For each selected object, you get the following:

  - An indication as to whether moving the object is possible.
  - A list of the affected information models, that is, the information models that reference the selected object.

  You can then refine the list of object that are eligible for re-factoring, and clear some of them.

- Another way is to select one or several packages as a whole. The selected packages will be moved to another package, becoming children packages of the target package.

  In this case, the system will only re-factor (move) the packages in which all objects can be moved.

  > Note:
  > This step means that if there is an issue that prevents moving one object in a package, the entire package cannot be moved.

  You can also refine the list of packages that are eligible for re-factoring, and clear some of them.

  A sub-package can also be moved to the *Content* folder (the root node for all packages).

Moving Information Models between Packages

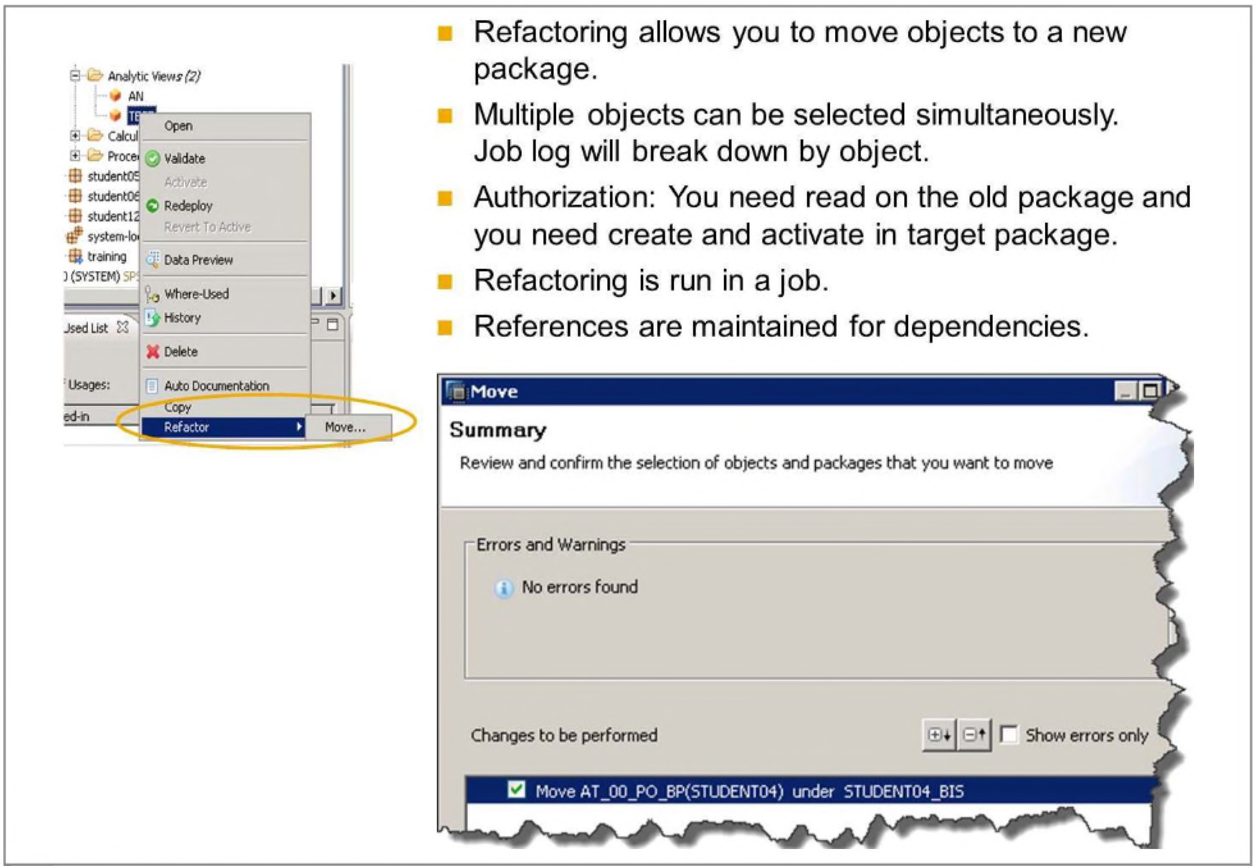


Figure 358: Moving Information Models Between Packages

You cannot re-factor an Information Model that has been modified and saved (but not activated) by another user. That is, an object that has an inactive version belonging to another user.

Before doing that, you must take over this information object, as you would do if you need to continue modifying this object.

Limitations to the Mass Copy and Re-factor Functionality

When moving (Refactor - Move) or copying (Mass Copy) information models from one package to another, and when renaming information models (Refactor - Rename), there are some important limitations that you must keep in mind.

- The SQL Syntax of procedures and script-based Calculation Views is never adjusted. If this syntax references directly a column view of the *_SYS_BIC* schema, it is not adjusted automatically.
- If an Attribute, Analytic or Graphical Calculation View references directly a column view of the *_SYS_BIC* schema (instead of the design-time object), the reference is not adjusted automatically.

Taking Over an Information Model - Switch Ownership

When you maintain information models (import, move, delete, and so on), there can be cases where a model is currently inactive due to the modification of another user. This sometimes prevents you from doing the maintenance action you want.

In this situation, you can use the *Takeover* feature.

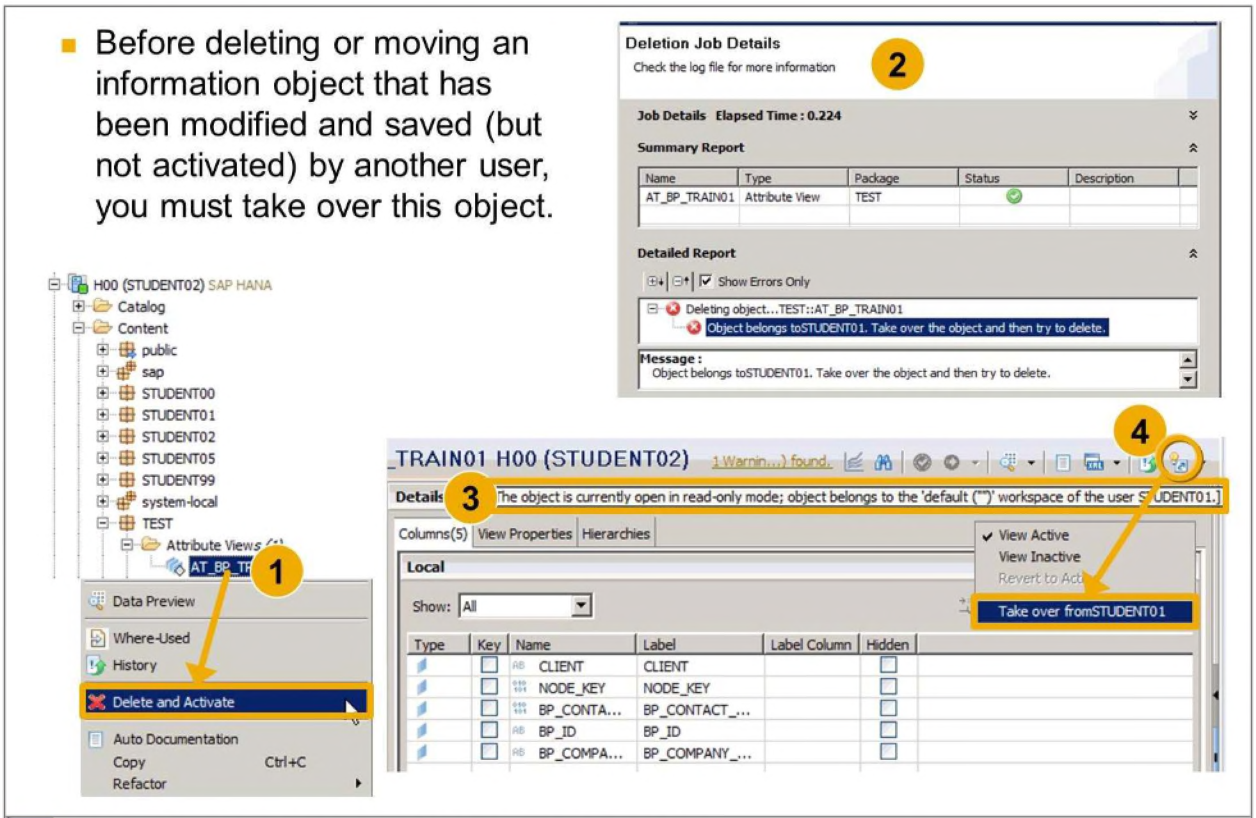


Figure 359: Taking Over a Content Object from Another User

> **Note:**
>
> - The takeover does not cancel the modifications made by the other user. Takeover actually enables you to become the owner of the inactive object.
> - Takeover only changes the owner of the inactive object. The active version of the information model is always owned by last the user who has activated it.

Switch Ownership of Inactive Objects

It is also possible to modify the ownership of several inactive objects owned by the same user in one single operation with the command **`Switch Ownership`**.

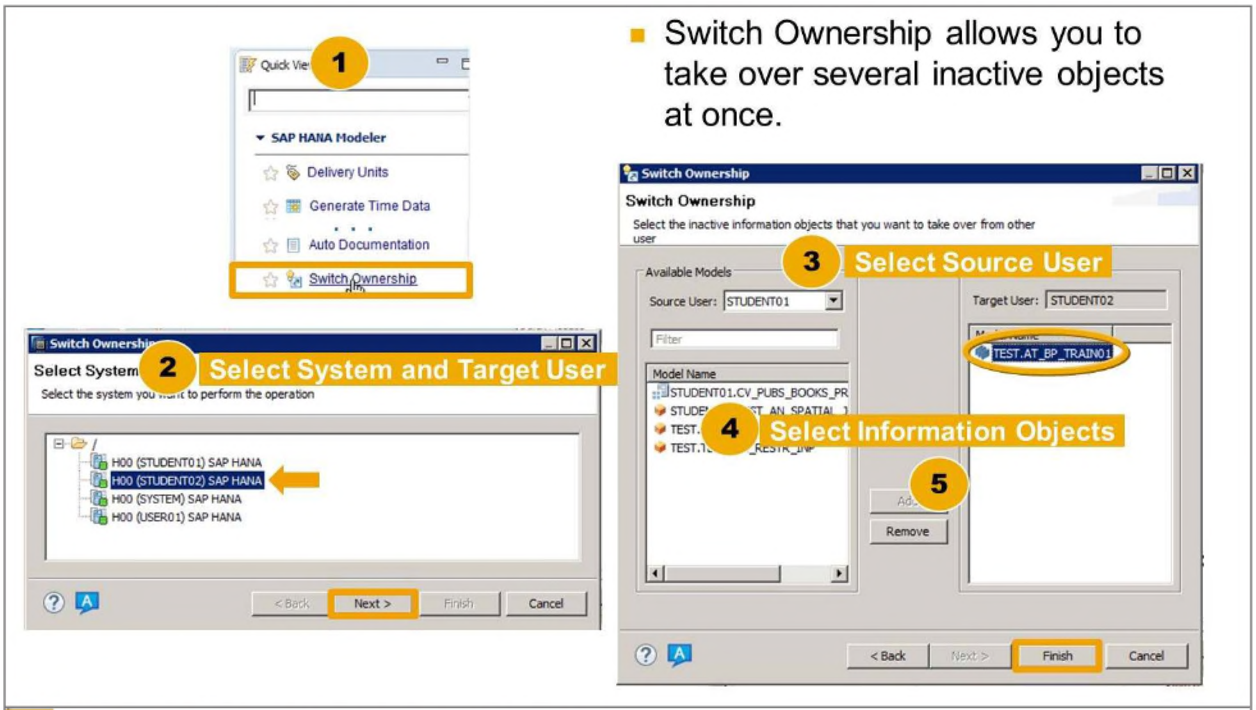


Figure 360: Switch Ownership of Inactive Objects

> Note:
> Compared with *Take Over*, the *Switch Ownership* feature also enables you to take over an inactive object that has never been activated.
>
> An information model that has never been activated is only visible to its owner, so another user cannot open it from the *Systems* view.

Translating Metadata Texts for Views

It is possible to maintain a number of view elements in several languages.

Translatable View Labels

- View description
- View nodes
- Elements (columns labels)
- Hierarchies
- Variables and Input Parameters

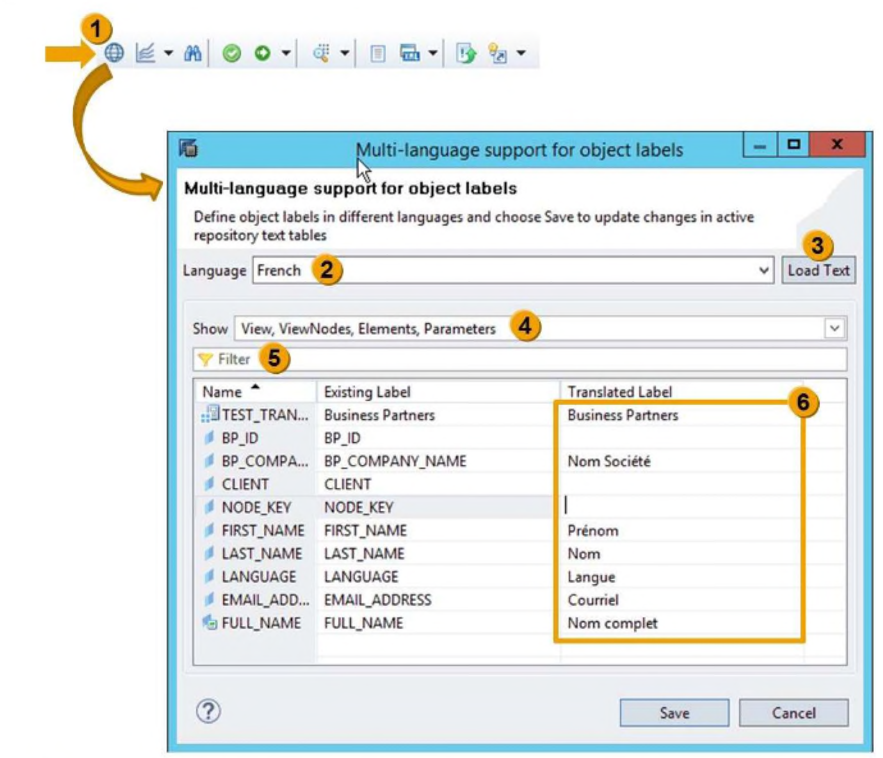


Figure 361: Translating View Labels

Note:
Before translating the metadata of an information view, you must:

- Active the *Translate* property of the view in the Semantics (*Properties* tab).
- Activate the information view.

Unit 6
Exercise 27

Define a Schema Mapping

Exercise Objectives

After completing this exercise, you will be able to:

- Simulate the transport of an information model from a system to another.
- Define a schema mapping.

Business Example

You are in charge of transporting information models between a Development system (DEV) and a Quality Assurance system (QA).

These information models reference some tables that are located in schemas that do not have the same name in both systems.

You must define a schema mapping on the target system so that your information models still work after they have been transported.

> Note:
> In your training landscape, you only have one SAP HANA system. So, during this exercise, you will only simulate the transport of the view by deleting the DEV schema and creating a QA schema.
>
> The two letters ## represent the two-digit group number assigned to you by the instructor.

Task 1: Create a Table in a New Schema and Build an Information Model

You will create a new schema, containing a table of Business Partners (BP), and then build a DIMENSION Calculation View referencing this table.

> Caution:
> In all the exercise, you must always replace **##** with your assigned group number, in particular in any SQL statement.

1. Open a SQL console in SAP HANA Studio.
2. From your *Student* folder, open the SQL file *HA300* → *Define_a_Schema_Mapping.sql*. Copy and paste all its content to the SQL Console.

   If credentials are required to access this shared folder, use the following:

Field	Value
Network Share User	`hanastudent`

Field	Value
Network Share Password	**hanareadonly**

3. In the SQL Console, replace any occurrence of **##** in the statements with your assigned group number.

> Caution:
> Until the end of this exercise, please make sure you execute only the SQL statements specified at each step. Do NOT execute the entire set of instructions from the SQL Console.

4. Create a new schema DEV_SCHEMA_## and a table BP, which is a copy of the SNWD_BP table from the EPM_MODEL schema.

   Execute the following statements (under TASK 1.a in your SQL Console) and check their correct execution.

```
CREATE SCHEMA DEV_SCHEMA_##;
CREATE TABLE DEV_SCHEMA_##.BP LIKE EPM_MODEL.SNWD_BP WITH DATA;
```

5. Create a simple DIMENSION Calculation View *CVD_BP2_##* in your package *STUDENT##*, with the following properties:

Property	Value
Name	**CVD_BP2_##**
Label	**Business Partners**
Source table (in the *Projection* node)	*DEV_SCHEMA_##.BP*
Output fields	• *CLIENT* • *NODE_KEY* • *BP_ID* • *BP_COMPANY_NAME*
Key attributes	• *CLIENT* • *NODE_KEY*

6. Save and Activate the Calculation View. What do you notice?
7. To be able to activate your model, which uses a table from the schema *DEV_SCHEMA_##*, execute the following SQL statement (under TASK 1.b in your SQL Console):

```
GRANT SELECT on SCHEMA DEV_SCHEMA_## TO _SYS_REPO WITH GRANT OPTION;
```

8. Activate the Calculation View again. What do you notice?
9. Close the *CVD_BP2_##* calculation view.

Task 2: Delete the DEV Schema and Observe the Impact on the Calculation View

1. Drop the schema *DEV_SCHEMA_##*.

   Execute the following statement (under TASK 2.a in your SQL Console)

```
DROP SCHEMA DEV_SCHEMA_## CASCADE;
```

2. Open the *CVD_BP2_##* calculation view. What do you notice?

> Note:
> You would get exactly the same result after transporting your Calculation View to another system where the schema *DEV_SCHEMA_##* is missing, or it exits but does not contain the *BP* table.
>
> Additionally, if you tried to activate the Calculation View (for example, when importing the Delivery Unit), you would get an error (`Table BP cannot be found or user has insufficient privileges`).

3. Close the *CVD_BP2_##* calculation view.

Task 3: Create a QA Schema and Define a Schema Mapping

1. Create a new schema *QA_SCHEMA_##*, which contains the same table *BP*. Grant the relevant authorizations to user _SYS_REPO to activate information models that reference tables from this schema.

   Execute the following statements (under TASK 3.a in your SQL Console).

```
CREATE SCHEMA QA_SCHEMA_##;
CREATE TABLE QA_SCHEMA_##.BP LIKE EPM_MODEL.SNWD_BP WITH DATA;
GRANT SELECT ON SCHEMA QA_SCHEMA_## TO _SYS_REPO WITH GRANT OPTION;
```

2. Define a schema mapping with the following properties:

Authoring Schema	Physical Schema
DEV_SCHEMA_##	*QA_SCHEMA_##*

> Note:
> The authoring schema must be entered manually, and it is case-sensitive. Make sure you enter it in upper case.

3. Open the *CVD_BP2_##* calculation view again, and check that it opens without any error.

4. What is the source schema displayed for table *BP* in the *Projection* node?

5. What is the *Schema* property of table *BP* in the *Properties* view? What does it mean?

Note:
The *Schema* property of the *BP* table can be modified manually in the *Properties* view. You have two options:

- You can specify another authoring schema that has a mapping defined to the physical schema *QA_SCHEMA_##* (there is not any other one in this training landscape).
- You can also specify the *QA_SCHEMA_##* physical schema itself, which means that you remove any reference to a different authoring schema.

Unit 6
Solution 27

Define a Schema Mapping

Exercise Objectives

After completing this exercise, you will be able to:

- Simulate the transport of an information model from a system to another.
- Define a schema mapping.

Business Example

You are in charge of transporting information models between a Development system (DEV) and a Quality Assurance system (QA).

These information models reference some tables that are located in schemas that do not have the same name in both systems.

You must define a schema mapping on the target system so that your information models still work after they have been transported.

> Note:
> In your training landscape, you only have one SAP HANA system. So, during this exercise, you will only simulate the transport of the view by deleting the DEV schema and creating a QA schema.
>
> The two letters ## represent the two-digit group number assigned to you by the instructor.

Task 1: Create a Table in a New Schema and Build an Information Model

You will create a new schema, containing a table of Business Partners (BP), and then build a DIMENSION Calculation View referencing this table.

> Caution:
> In all the exercise, you must always replace **##** with your assigned group number, in particular in any SQL statement.

1. Open a SQL console in SAP HANA Studio.

   a) In the *Systems* view, right-click your *SAP HANA System H00 (STUDENT##)* and choose *Open SQL Console*.

2. From your *Student* folder, open the SQL file *HA300* → *Define_a_Schema_Mapping.sql*. Copy and paste all its content to the SQL Console.

   If credentials are required to access this shared folder, use the following:

Field	Value
Network Share User	`hanastudent`

Field	Value
Network Share Password	**hanareadonly**

a) Choose *Start → Student - hanastudent - hanareadonly*.

> Note:
> If the Student shortcut is not pinned to your Windows *Start* page, use the *Search* tool (magnifier on the top-right corner or the *Start* page).

b) If needed, enter the provided credentials, and choose *OK*.

c) Open the file *HA300 → Define_a_Schema_Mapping.sql*.

d) Select all the file's content (you can press Ctrl+A) and copy it to the clipboard (press Ctrl+C).

e) To paste the content in the SQL Console, press Shift+Ctrl+INS (a special shortcut is used in the SQL Console) or right-click anywhere inside the SQL Console and choose *Paste*.

3. In the SQL Console, replace any occurrence of **##** in the statements with your assigned group number.

a) Press Ctrl+F (Find/Replace).

b) In the *Find* field, enter **##**.

c) In the *Replace with* field, enter your group number.

d) Choose *Replace All*.

> Caution:
> Until the end of this exercise, please make sure you execute only the SQL statements specified at each step. Do NOT execute the entire set of instructions from the SQL Console.

4. Create a new schema DEV_SCHEMA_## and a table BP, which is a copy of the SNWD_BP table from the EPM_MODEL schema.

Execute the following statements (under TASK 1.a in your SQL Console) and check their correct execution.

```
CREATE SCHEMA DEV_SCHEMA_##;
CREATE TABLE DEV_SCHEMA_##.BP LIKE EPM_MODEL.SNWD_BP WITH DATA;
```

a) Highlight the specified statements and choose *Execute* (press F8).

b) Check that the execution was successful.

5. Create a simple DIMENSION Calculation View *CVD_BP2_##* in your package *STUDENT##*, with the following properties:

Property	Value
Name	**CVD_BP2_##**

Property	Value
Label	**Business Partners**
Source table (in the *Projection* node)	*DEV_SCHEMA_##.BP*
Output fields	• *CLIENT* • *NODE_KEY* • *BP_ID* • *BP_COMPANY_NAME*
Key attributes	• *CLIENT* • *NODE_KEY*

a) In the Content folder, right-click your *STUDENT##* package and choose *New → Calculation View*.

b) Enter the Calculation View name and label as specified in the table. In the *Data Category* dropdown list, choose *DIMENSION*.

c) In the *Scenario* pane, choose the + sign on the right of the *Projection* node.

d) In the *Search* field, enter **BP**, select the *BP (DEV_SCHEMA_##)* table and choose *OK*.

e) In the *Details* pane of the *Projection* node, right-click the *CLIENT* column and choose *Add To Output*.

f) Repeat the previous step for the other fields to be included in the output.

g) In the *Semantics* node, select the *Key* checkbox for columns *CLIENT* and *NODE_KEY*.

6. Save and Activate the Calculation View. What do you notice?

a) Choose *Save and Activate*.
The Calculation View cannot be validated. The error details mention: `(...) User is not authorized`.

> Note:
> This is the typical error that you get when _SYS_REPO, the technical user who is in charge of activating the information models, cannot execute a `SELECT` statement on the schema that contains the source tables referenced by the information model.

7. To be able to activate your model, which uses a table from the schema *DEV_SCHEMA_##*, execute the following SQL statement (under TASK 1.b in your SQL Console):

```
GRANT SELECT on SCHEMA DEV_SCHEMA_## TO _SYS_REPO WITH GRANT OPTION;
```

a) Highlight the specified statement and choose *Execute* (press F8).

b) Check that the execution was successful.

8. Activate the Calculation View again. What do you notice?

a) Choose *Save and Activate*.
   This time, the Calculation View activation is successful.

9. Close the *CVD_BP2_##* calculation view.

Task 2: Delete the DEV Schema and Observe the Impact on the Calculation View

1. Drop the schema *DEV_SCHEMA_##*.
   Execute the following statement (under TASK 2.a in your SQL Console)

```
DROP SCHEMA DEV_SCHEMA_## CASCADE;
```

   a) Highlight the specified statements and choose *Execute* (press F8).

   b) Check that the execution was successful.

2. Open the *CVD_BP2_##* calculation view. What do you notice?

   a) In the *Systems* view, right-click the Calculation View *STUDENT## → CVD_BP2_##* and choose *Open*.
      The data source of the *Projection* node (the table *BP* from schema *DEV_SCHEMA_##*) has an icon with a red dot, meaning that the table is no longer available.

> Note:
> You would get exactly the same result after transporting your Calculation View to another system where the schema *DEV_SCHEMA_##* is missing, or it exits but does not contain the *BP* table.
>
> Additionally, if you tried to activate the Calculation View (for example, when importing the Delivery Unit), you would get an error (`Table BP cannot be found or user has insufficient privileges`).

3. Close the *CVD_BP2_##* calculation view.

Task 3: Create a QA Schema and Define a Schema Mapping

1. Create a new schema *QA_SCHEMA_##*, which contains the same table *BP*. Grant the relevant authorizations to user _SYS_REPO to activate information models that reference tables from this schema.
   Execute the following statements (under TASK 3.a in your SQL Console).

```
CREATE SCHEMA QA_SCHEMA_##;
CREATE TABLE QA_SCHEMA_##.BP LIKE EPM_MODEL.SNWD_BP WITH DATA;
GRANT SELECT ON SCHEMA QA_SCHEMA_## TO _SYS_REPO WITH GRANT OPTION;
```

   a) Highlight the specified statements and choose *Execute* (press F8).

   b) Check that the execution was successful.

2. Define a schema mapping with the following properties:

Authoring Schema	Physical Schema
DEV_SCHEMA_##	*QA_SCHEMA_##*

> Note:
> The authoring schema must be entered manually, and it is case-sensitive. Make sure you enter it in upper case.

a) If needed, to display the *Quick View*, choose *Help* → *Quick View*.

b) In the *Quick View*, choose *Schema Mapping*.

c) Select the *H00 (STUDENT##)* system, and choose *Next*.

d) Define the mapping as specified in the table above, and choose *Finish*.

3. Open the *CVD_BP2_##* calculation view again, and check that it opens without any error.

a) In the Systems view, right-click the Calculation View *STUDENT##* → *CVD_BP2_##* and choose *Open*.
In the *Projection* node, the data source (table *BP*) is now available.

4. What is the source schema displayed for table *BP* in the *Projection* node?

a) In the *Scenario* pane, select the *Projection* node.
The source schema for table *BP* is *QA_SCHEMA_##*.

5. What is the *Schema* property of table *BP* in the *Properties* view? What does it mean?

a) In the *Details* pane of the *Projection* node, select the *"QA_SCHEMA_##".BP* table.

b) Observe the information displayed in the *Properties* view.
The *Schema* property is *DEV_SCHEMA_## (QA_SCHEMA_##)*. This means that the designed view used the table *BP* from the schema *DEV_SCHEMA_##*, but due to a schema mapping, this data is actually read from the table *BP* of the schema *QA_SCHEMA_##*.

> Note:
> The *Schema* property of the *BP* table can be modified manually in the *Properties* view. You have two options:
>
> - You can specify another authoring schema that has a mapping defined to the physical schema *QA_SCHEMA_##* (there is not any other one in this training landscape).
> - You can also specify the *QA_SCHEMA_##* physical schema itself, which means that you remove any reference to a different authoring schema.

LESSON SUMMARY

You should now be able to:

- Manage modeling content
- Define package-specific authoring schema
- Transport information models between two systems
- Move and copy information models within a system
- Take over an information model edited by another user
- Translate metadata texts

Unit 6
Lesson 3

Converting Deprecated Information Objects

LESSON OVERVIEW

In this lesson, you will learn how to use the migration functionality offered by SAP HANA to migrate Attribute and Analytic Views into Calculation Views, and script-based Calculation Views into Table Functions. This migration functionality has been introduced with SAP HANA SPS11.

Business Example

You work on a customer project where SAP HANA has been upgraded to SPS12.

You need to know how to convert your existing Information Views into Calculation Views and replace scripted Calculation Views with Table Functions.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Convert Attribute and Analytic Views into Calculation Views
- Convert Scripted Calculation Views into Table Functions

Migration Utility for Attribute and Analytic Views

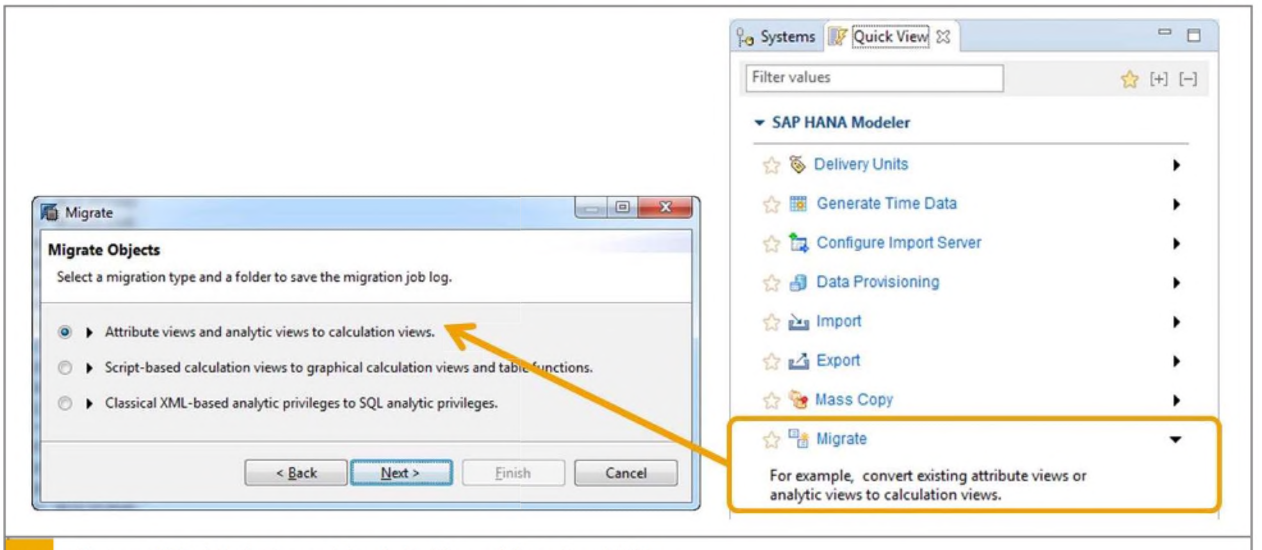


Figure 362: Attribute and Analytic Views Migration Utility

From SPS11 onwards, SAP HANA provides a specific utility to migrate the following deprecated Information Models types:

- Attribute and Analytic Views
- Scripted Calculation Views

- Classic Analytic Privileges

In this lesson, we will focus on Attribute and Analytic Views, as well as scripted Calculation Views.

> Note:
> A number or considerations about how to use the migration utility are common to the two other types of modeling artifacts. In particular, we will discuss some important best practices to make the migration easier and avoid common issues.

How to Migrate Attribute and Analytic Views into Calculation Views

1. Start the Migration Utility
   a) In the *Quick View*, choose *Migrate*.
2. Choose the type of object *Attribute Views and Analytic Views...* and choose *Next*.
   Optionally, you can decide if you want to generate a migration log file (and specify its location).
3. Choose the views you want to migrate.
   You can select individual objects or the entire packages.
4. Decide if you want to migrate and replace the information views, which is the default behavior, or simulate the migration by creating copies, to first test and validate the migrated objects.
   If you want to work in simulation mode, select the *Copy and Migrate* checkbox.

   > Note:
   > - The references between the migrated objects will always be adjusted.
   > - The references to migrated objects included in objects that you do not migrate are also adjusted, except if you choose to *Copy and Migrate*.

5. Review the List of Impacted Objects.
   This list includes all the information views in which references to migrated objects will be automatically adjusted, and identifies objects that you will have to migrate during a next migration round.
6. Run the migration job and check the log (if any).
   a) Choose *Finish*.
   b) Open the log file and review the results.

> Note:
> At this stage, for each migrated object, the workspace contains two inactive versions of this object:
>
> - The former Attribute or Analytic View
> - The new Calculation View

7. Activate the migrated objects, which includes deleting the old version of the objects (except in a "simulated" migration).

   Alternatively, you can roll back the migration by deleting the inactive objects and revert to the last active version of these objects.

   a) In the *Quick View*, choose *Activate*.

   b) Check the list of inactive objects in the right pane, and choose *Finish*.

   c) Alternatively, to roll back the migration, in the *Quick View*, choose *Delete Inactive Objects*, check the list of objects and choose *Finish*.

Migration Log

A migration log records the main changes that the utility performs during the migration activity and provides information on the migration status.

Main Information Provided by the Migration Log

- Total number of objects selected for migration.
- Total number of objects successfully converted to new object types.
- List of objects that require user actions before activation.
- List of impacted objects whose references have been adjusted automatically during migration.

  The impacted objects are also converted to the new object type.
- List of impacted objects whose references could not be adjusted automatically.

  You will have to manually select such impacted objects and convert them to the new object types in a new migration step.

General Considerations before a Migration

Object Dependencies and Package Structure

The dependencies between Information Views must be considered carefully during the migration.

Object Dependencies and Package Structure

- All the Attribute Views referenced by an Analytic View must be migrated before the (migrated) Analytic View can be successfully activated.
- A migrated Attribute View (that is, a DIMENSION Calculation View) can be successfully activated only when all the Analytic Views referencing it have been migrated.

- The references to migrated objects in Calculation Views are automatically adjusted during activation, even if the packages containing the Calculation Views are not selected for migration.

Step by Step Migration Process

To allow a staggered migration, you can apply the following recommendations:

Migration Process Recommendation

- Several migration steps can be executed to reach a consistent state of all migrated objects.
- Then all migrated objects can get activated at once.
- This allows you to cancel the migration and revert to the previous status, by choosing *Delete inactive objects* from the *Quick View*.

Handling Functional Differences between Attribute/Analytic Views and Calculation Views

To avoid losing some features from the source Information Views, apply the following recommendations:

Features with Different Behavior in Calculation Views

- Hidden columns can be made visible during migration (dedicated option in the migration utility), or they will be missing in the resulting Calculation View.
- The internal *row.count* column that existed in every Analytic View is made visible in the resulting Calculation View, as the *Count Star Column* in its properties.
- A derived Attribute View is converted into a standalone DIMENSION Calculation View, with no dependency to the main view anymore.
- Column Filters in Attribute and Analytic Views are converted to *Filter Expressions* in the new Calculation Views
- Input Parameter-based Filters in the Data Foundation of Attribute and Analytic Views are converted into additional *Projection* nodes with *Filter Expression*.
- The *Relational Optimization* setting in Analytic Views is dropped during migration.

  Indeed, the general optimization for Calculation Views is applied.

> Caution:
> As of SPS12, SAP HANA does not provide in Calculation Views the support for Search Attributes, which are only available in Attribute Views.

How to Secure the Migration

Key Recommendations for the Migration Process

- Take a back-up of all objects before executing the migration
- Start migration with a clean workspace, with all Information Views activated

> Note:
> If you have inactive objects in your workspace before the migration, they will be mixed up with the inactive migrated objects, which will make the migration check more difficult, and prevent a proper migration roll-back.

- Simulate the migration activity before migrating

  This is a very convenient way to check the impact of migration without affecting at all the source Information Views.

- Migrate Information Views before migrating Analytic Privileges

  If you plan to migrate both Attribute/Analytic Views to Calculation Views and Classical Analytic Privileges to SQL Analytic Privileges, the migration should be done in two distinct steps. Information Views first, then Analytic Privileges.

- Undoing migration changes is only possible before activating the objects

Unit 6
Exercise 28

Convert Attribute and Analytic Views to Calculation Views

Exercise Objectives

After completing this exercise, you will be able to:

- Use the *Mass Copy* functionality to copy information views and preserve the links between the copied ones
- Move information models from one package to another
- Convert Attribute Views and Analytic Views into Calculation View
- Rename information models

Business Example

As a view modeler on SAP HANA, you are involved in a project to migrate from a previous version of SAP HANA to SAP HANA SPS11.

You need to understand how to leverage the new migration function on your existing Attribute and Analytic Views to harmonize the modeling approach to Calculation Views only.

You will work with an Analytic View *AN_PO* and three related Attribute Views, *AT_BP*, *AT_CURRENCY* and *AT_PD*.

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.

Task 1: Copy the Source Information Views to your Package

In this training environment, you will create your own copy of the sources information views from templates, so that you do not interfere with other students.

You will use the *Mass Copy* functionality to make your personal copy of the template information views to be migrated later on.

> Caution:
> You must use the *Mass Copy* functionality to preserve the reference between the models you copy.
>
> A simple copy and paste would create copies of each object, but the copy of the Analytic View would still reference the source Attribute Views (not the copies).

1. Check the views that you need to migrate in the source package.

The information views to migrate are the following:

Source package	Information Views to migrate	Destination package
training → ha300	• *AT_BP* • *AT_CURRENCY* • *AT_PD* • *AN_PO*	*STUDENT##*

2. From the *Quick View*, start the *Mass Copy* assistant and choose or confirm the system on which you want to work in and the user who will perform the mass copy.

> Note:
> This is required when you work on several systems referenced in SAP HANA Studio and also when you are connected to the same system with different users.

3. Define the package mapping for your Mass Copy, based on the Source and Destination packages mentioned above.

   For the source, you can only select a child package from the root, not a sub-package. In this case, the *training* package.

> Note:
>
> - The package mappings can be saved in your local workspace or in the SAP HANA system, so that you can reuse them later on.
> - In the context of this training, you do not want to share this mapping definition with the other users, so you will store it only in your local workspace.

4. Select the models to copy according to the table presented in the first step.
5. In the *Summary* view, check the status, source and target package for each model to copy and execute the Mass Copy.

   Note that the copy will create a sub-package *ha300* in the destination folder.
6. In the *Job Log* view, check that the Mass Copy was executed successfully.
7. Activate the copied information models in your package *STUDENT## → ha300* and check the activation log.

> Caution:
> Make sure you select only the four information models that you are migrating from your *STUDENT##.ha300* package.

Task 2: Move the Copied Information Models to another Package

As part of this training, you want to execute the migration of the information models in a dedicated package. This is an opportunity to see how you can move information models between packages.

1. Create a new package *MIGRATED* as a child of your STUDENT## package.
2. Move the information models you have just copied. The target package is *STUDENT## → MIGRATED*.

> Hint:
> In the *Systems* view, you must use the command *Refactor → Move* from the context menu of the information models.

Task 3: Migrate Information Views

You are now ready to convert your Attribute and Analytic Views into Calculation views.

1. Migrate the views all at once with the Migration assistant from the *Quick View*.

   You will migrate the information objects within the same package *STUDENT## → MIGRATED*, that is, you will not use the *Copy and migrate* option.

   You also want to generate a log file.

2. Activate the new Calculation Views and review the activation log.

   The scope of the activation must also include the source Attribute and Analytic Views, which must be deleted.

3. Rename the new information models as follows, to apply the relevant naming convention for Calculation Views.

> Hint:
> Use the context menu in the *Systems* view and choose *Refactor → Rename*

View to rename	New name
AN_PO	*CVCS_PO*
AT_BP	*CVD_BP*
AT_CURRENCY	*CVD_CURRENCY*
AT_PD	*CVD_PD*

   After renaming, the workspace contains the four inactive renamed Calculation Views, which are visible, but also the objects with their old names. These also need to be activated in order to remove them completely from the workspace.

4. Activate the new Information models, and trigger the deletion of the ones with the original package names.

   This can be done in a single operation, with the context menu of the *Systems* view or from the *Quick View*.

5. Delete the package *STUDENT## → ha300*.

Make sure the package is empty before you delete it.

Unit 6
Solution 28

Convert Attribute and Analytic Views to Calculation Views

Exercise Objectives

After completing this exercise, you will be able to:

- Use the *Mass Copy* functionality to copy information views and preserve the links between the copied ones
- Move information models from one package to another
- Convert Attribute Views and Analytic Views into Calculation View
- Rename information models

Business Example

As a view modeler on SAP HANA, you are involved in a project to migrate from a previous version of SAP HANA to SAP HANA SPS11.

You need to understand how to leverage the new migration function on your existing Attribute and Analytic Views to harmonize the modeling approach to Calculation Views only.

You will work with an Analytic View *AN_PO* and three related Attribute Views, *AT_BP*, *AT_CURRENCY* and *AT_PD*.

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.

Task 1: Copy the Source Information Views to your Package

In this training environment, you will create your own copy of the sources information views from templates, so that you do not interfere with other students.

You will use the *Mass Copy* functionality to make your personal copy of the template information views to be migrated later on.

> Caution:
> You must use the *Mass Copy* functionality to preserve the reference between the models you copy.
>
> A simple copy and paste would create copies of each object, but the copy of the Analytic View would still reference the source Attribute Views (not the copies).

1. Check the views that you need to migrate in the source package.

The information views to migrate are the following:

Source package	Information Views to migrate	Destination package
training → ha300	• *AT_BP* • *AT_CURRENCY* • *AT_PD* • *AN_PO*	*STUDENT##*

a) In the *Systems* view, browse the *Content* folder to display the folder *training → ha300*.

b) Make sure that the four views mentioned in the table are available.

2. From the *Quick View*, start the *Mass Copy* assistant and choose or confirm the system on which you want to work in and the user who will perform the mass copy.

> Note:
> This is required when you work on several systems referenced in SAP HANA Studio and also when you are connected to the same system with different users.

a) Choose *Help → Quick View*.

b) Choose *Mass Copy*.

c) In the *Select System* window, choose the system/user pair *H00 (STUDENT##) SAP HANA*.

d) Choose *Next*.

3. Define the package mapping for your Mass Copy, based on the Source and Destination packages mentioned above.
For the source, you can only select a child package from the root, not a sub-package. In this case, the *training* package.

> Note:
>
> - The package mappings can be saved in your local workspace or in the SAP HANA system, so that you can reuse them later on.
> - In the context of this training, you do not want to share this mapping definition with the other users, so you will store it only in your local workspace.

a) In the *Manage Package Mappings* window, in the *Store Package Mappings in* option box, select the *Local Workspace* radio button.

b) Select the first row, and select the source package *training* and the target package *STUDENT##*.

> Note:
> If the *training* package is not available in the *Source package* dropdown list, you must first remove the global mapping defined for this package.

c) Choose *Next*.

4. Select the models to copy according to the table presented in the first step.

a) In the *Available* pane, expand the package tree and select the *training → ha300 → AT_BP* Attribute View.

> Hint:
> You can press and hold the Ctrl key to select several views at the same time.

b) Choose *Add*.

c) If needed, repeat the same steps for the other views to copy.

d) Choose *Next*.

5. In the *Summary* view, check the status, source and target package for each model to copy and execute the Mass Copy.

Note that the copy will create a sub-package *ha300* in the destination folder.

a) Check the columns *status*, *source* and *target package* and *message* for each package.

b) Choose *Finish*.

6. In the *Job Log* view, check that the Mass Copy was executed successfully.

a) In the *Job Log* view, check that the status of the last *Copy Models* operation is *Completed Successfully*.

7. Activate the copied information models in your package *STUDENT## → ha300* and check the activation log.

> Caution:
> Make sure you select only the four information models that you are migrating from your *STUDENT##.ha300* package.

a) In the *Systems* view, right-click the package *STUDENT## → ha300* and choose *Activate*.

If needed, remove any other information view than the four ones you are migrating from the *Selected* pane.

b) Alternatively, you can multi-select the four models by pressing Ctrl while you choose them, and then right-click the selection and choose *Activate*.

Check the list of models in the *Selected* pane.

c) Choose *Activate*.

d) In the *Job Log* view, check that the status of the last *Activation* job is *Completed successfully*.

Task 2: Move the Copied Information Models to another Package

As part of this training, you want to execute the migration of the information models in a dedicated package. This is an opportunity to see how you can move information models between packages.

1. Create a new package *MIGRATED* as a child of your STUDENT## package.

   a) Right-click your package *STUDENT##* and select *New* → *Package...*.

   b) Enter the package name **`STUDENT##.MIGRATED`** (make sure you do not modify the parent package name *STUDENT##*) and choose *OK*.

2. Move the information models you have just copied. The target package is *STUDENT##* → *MIGRATED*.

> Hint:
> In the *Systems* view, you must use the command *Refactor* → *Move* from the context menu of the information models.

   a) In the *Systems* view, right-click the *STUDENT00* → *ha300* → *AT_BP* view and choose *Refactor* → *Move*.

> Note:
> Optionally, you can select all the models by pressing the Ctrl key while selecting them, and move them all in one operation.

   b) In the *Select Destination* screen, select the *STUDENT##* → *MIGRATED* package and choose *Next*.

   c) Review the *Summary* dialog box and choose *Finish*.

   d) If needed, repeat the same steps for the three other models.

Task 3: Migrate Information Views

You are now ready to convert your Attribute and Analytic Views into Calculation views.

1. Migrate the views all at once with the Migration assistant from the *Quick View*.

   You will migrate the information objects within the same package *STUDENT##* → *MIGRATED*, that is, you will not use the *Copy and migrate* option.

   You also want to generate a log file.

   a) In the *Quick View*, choose *Migrate*.

   b) For the migration type, select *Attribute views and analytic views to calculation views*.

   c) If needed, select the *Create Migration Log* checkbox and keep the default Migration Log Path.

d) In the left pane of the *Select Packages* window, select the *STUDENT## → MIGRATED* package and choose *Add*.

e) Choose *Next*.

f) Review the list of impacted objects and choose *Finish*.

2. Activate the new Calculation Views and review the activation log.

   The scope of the activation must also include the source Attribute and Analytic Views, which must be deleted.

   a) In the *Quick View*, choose *Activate*.

   b) Select or confirm your system and user and choose *Next*.
   In the *Activate* window, all the inactive objects, including deleted ones, are selected by default.

   c) Make sure the list of selected objects is relevant.

   You should only see the four new Calculation Views and the four source Attribute or Analytic Views.

   d) Choose *Finish*.

   e) In the *Job Log* view, check that the status of the last *Activation* task is *Completed Successfully*.

   f) Optionally, double-click the task to view the detailed log.

3. Rename the new information models as follows, to apply the relevant naming convention for Calculation Views.

Hint:
Use the context menu in the *Systems* view and choose *Refactor → Rename*

View to rename	New name
AN_PO	*CVCS_PO*
AT_BP	*CVD_BP*
AT_CURRENCY	*CVD_CURRENCY*
AT_PD	*CVD_PD*

   a) In the *Systems* view, right-click the view *STUDENT## → MIGRATED → AN_PO* and choose *Refactor → Rename*.

   b) Enter the new name **CVCS_PO** and choose *Next*.

   c) Check the summary for warnings and errors, and confirm the list of dependent objects that will be impacted by the name change.

   d) Choose *Finish*.

e) Right-click the *STUDENT##>MIGRATED* package and choose *Activate*. Make sure the *Selected* pane only contains the *AN_PO[Deleted]* and *CVCS_PO* views, and choose *Activate*.

f) Rename the three migrated attribute views *AT_BP*, *AT_CURRENCY* and *AT_PD* in a similar way.

> Note:
> You do not need to activate these views one by one at this stage.

After renaming, the workspace contains the four inactive renamed Calculation Views, which are visible, but also the objects with their old names. These also need to be activated in order to remove them completely from the workspace.

4. Activate the new Information models, and trigger the deletion of the ones with the original package names.

   This can be done in a single operation, with the context menu of the *Systems* view or from the *Quick View*.

   a) Right-click the *STUDENT## → MIGRATED* package and choose `Activate`.

   b) Make sure seven models in total are selected in the right pane. The three former attribute views (to confirm deletion), the three new Dimension Calculation Views, and the *CVCS_PO* view that has been impacted by the renaming.

   c) Choose *Finish*.

5. Delete the package *STUDENT## → ha300*.

   Make sure the package is empty before you delete it.

   a) Right-click the package *STUDENT## → ha300* and choose *Delete*.

   b) Choose *OK*.

Migrating Scripted Calculation Views

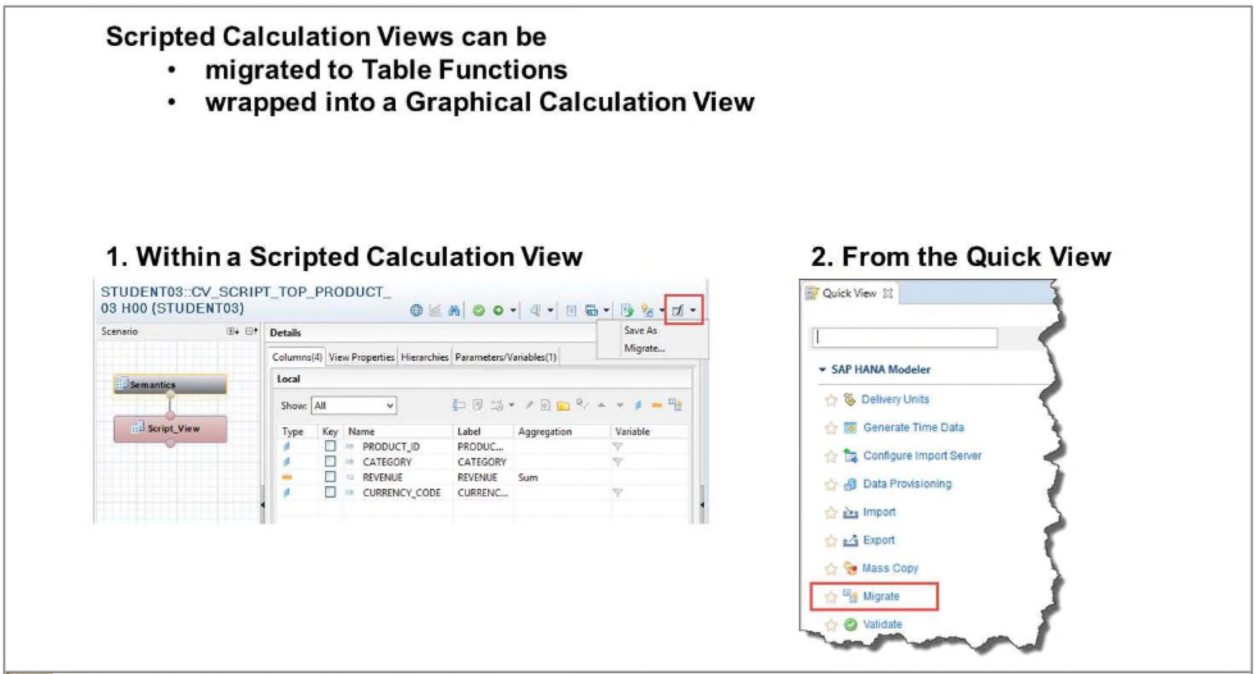


Figure 363: Migrating Scripted Calculation Views

Existing Script-based Calculations Views can be migrated to table functions, and these table functions can be used as data sources in Graphical Calculation Views. The migration utility can be accessed directly within the editor of a Scripted Calculation View or via the Migrate functionality available in the Quick View.

The Scripted Calculation View Migration Tool

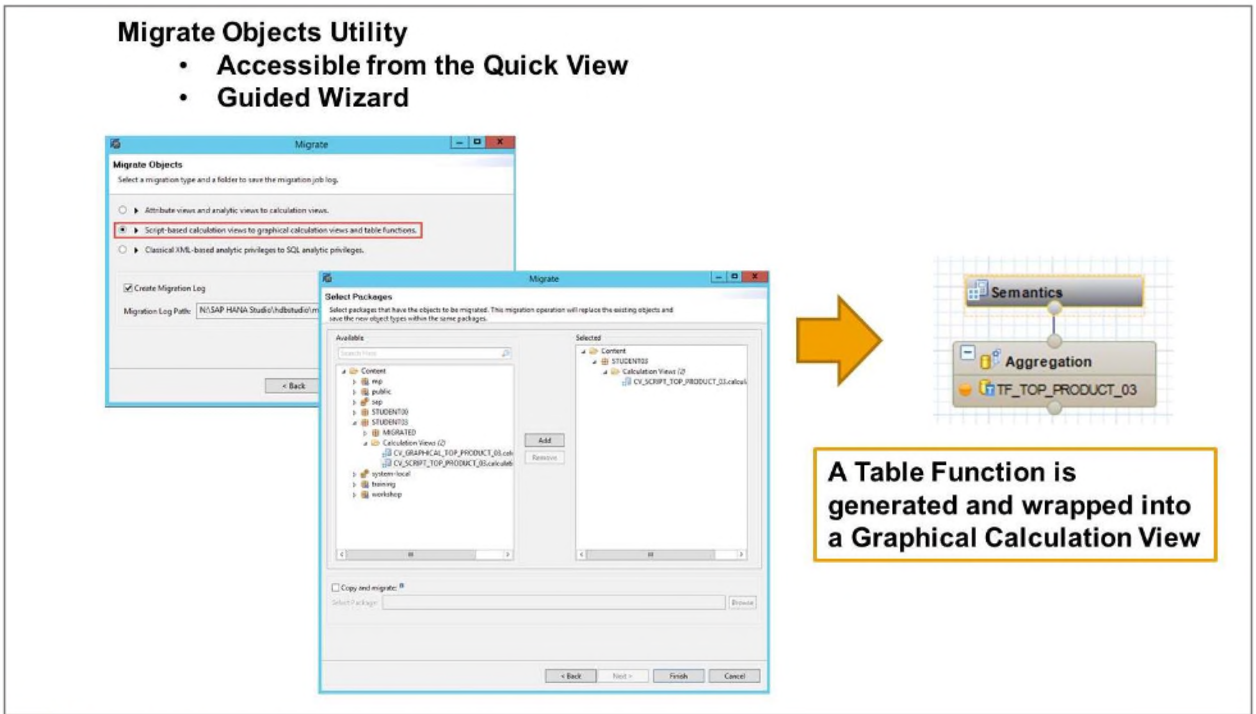


Figure 364: The Scripted Calculation View Migration Tool

Unit 6
Exercise 29

Migrate a Scripted Calculation View to a Table Function (Optional)

Exercise Objectives

After completing this exercise, you will be able to:

- Convert a scripted Calculation View to a Table User Defined Function (UDF)
- Wrap the Table UDF in a Graphical Calculation View

> Note:
> From SAP HANA SPS11 onwards, you will not create scripted Calculation Views anymore, but you might need to migrate (convert) existing ones in older systems to the new Table Functions.

Business Example

In your SAP HANA system, a scripted Calculation View has been used in the past to define a complex logic. The logic retrieves the list of the top 10 products, in terms of revenue generated in the last N years, based on your sales orders data.

To catch up with the most recent modeling approaches and gain performance, you want to convert this scripted Calculation View into a Table Function and wrap it into a graphical Calculation View.

1. Copy the source calculation view *CV_SCRIPT_TOP_PRODUCTS* from the *training* → *ha300* package, to your *STUDENT##* package.

   Name the copied view **CV_SCRIPT_TOP_PRODUCTS_##**.

2. Open the copied view and review its definition, in particular the columns and input parameter.

3. Activate the Calculation View *CV_SCRIPT_TOP_PRODUCTS_##*.

4. Test your Calculation View, by performing a Data Preview from the HANA Studio. Use **5** as value for the input parameter.

5. Convert the scripted Calculation View into a Table Function embedded in a graphical Calculation View.

   Use the *Migrate* tool available in the *Quick View*.

6. Activate the generated Table Function.

> Hint:
> To see the Table Function in your *MIGRATED* package, you must select the *Show All Objects* checkbox in the preferences of SAP HANA Studio. This option is located in *SAP HANA → Modeler → Content Presentation*.

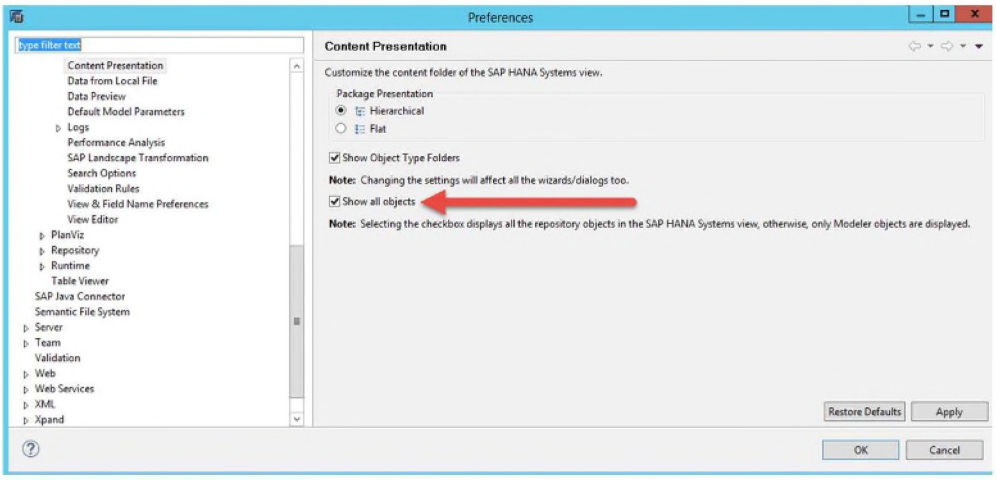


Figure 367: Show All Objects

7. Visualize the new graphical Calculation View in your *STUDENT## → MIGRATED* package and note that the generated Table Function is used as a data source for the graphical Calculation View.

8. Activate the new Calculation View

9. Perform a Data Preview from the generated graphical Calculation View in your *STUDENT##.MIGRATED* package.

   Follow the same steps as before, when previewing the data of for the scripted Calculation View, and ensure you achieve the same result in the *Raw Data* tab of the Data Preview.

Migrate a Scripted Calculation View to a Table Function (Optional)

Exercise Objectives

After completing this exercise, you will be able to:

- Convert a scripted Calculation View to a Table User Defined Function (UDF)
- Wrap the Table UDF in a Graphical Calculation View

> Note:
> From SAP HANA SPS11 onwards, you will not create scripted Calculation Views anymore, but you might need to migrate (convert) existing ones in older systems to the new Table Functions.

Business Example

In your SAP HANA system, a scripted Calculation View has been used in the past to define a complex logic. The logic retrieves the list of the top 10 products, in terms of revenue generated in the last N years, based on your sales orders data.

To catch up with the most recent modeling approaches and gain performance, you want to convert this scripted Calculation View into a Table Function and wrap it into a graphical Calculation View.

1. Copy the source calculation view *CV_SCRIPT_TOP_PRODUCTS* from the *training* → *ha300* package, to your *STUDENT##* package.

   Name the copied view **CV_SCRIPT_TOP_PRODUCTS_##**.

   a) In the *Systems* view, expand the *Content* → *training* → *ha300* folder. Right-click the Calculation View *CV_SCRIPT_TOP_PRODUCTS* and choose *Copy*.

   b) Right-click the package *STUDENT##* and choose *Paste*. Adjust the view name as per the instruction, and choose *OK*.

2. Open the copied view and review its definition, in particular the columns and input parameter.

3. Activate the Calculation View *CV_SCRIPT_TOP_PRODUCTS_##*.

   a) Click *Save and Activate*.

4. Test your Calculation View, by performing a Data Preview from the HANA Studio. Use **5** as value for the input parameter.

   a) Choose *Data Preview*.

b) Enter **5** as value for the input parameter (in the *From* field) and choose *OK*:

c) Choose the *Raw Data* tab and visualize the output of the Calculation View.

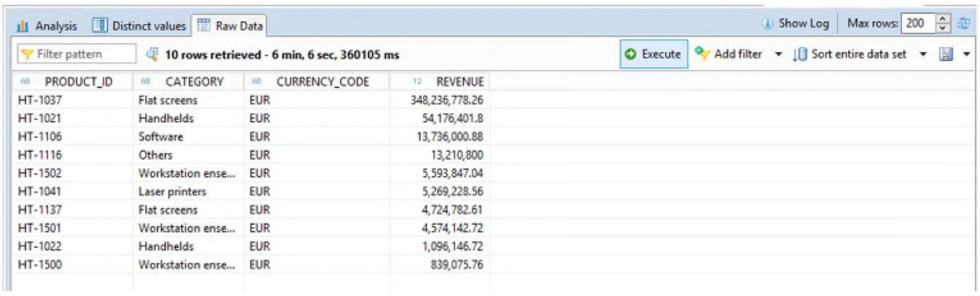


Figure 365: Scripted Calculation View - Raw Data

5. Convert the scripted Calculation View into a Table Function embedded in a graphical Calculation View.

   Use the *Migrate* tool available in the *Quick View*.

   a) Navigate to *Window → Show View → Other...* and select *Quick View*.

   b) In the *Quick View*, choose *MIGRATE*.

   c) Select your SAP HANA System and choose **`Next`**.

   d) Select the option to convert script-based calculation views to graphical calculation views and table functions and choose *Next*.

   e) Select your scripted Calculation View *CV_SCRIPT_TOP_PRODUCTS_##* and choose *Add*.

   f) Select the *Copy and Migrate* checkbox and select the *STUDENT##.MIGRATED* target package.

   g) Choose *Finish*.

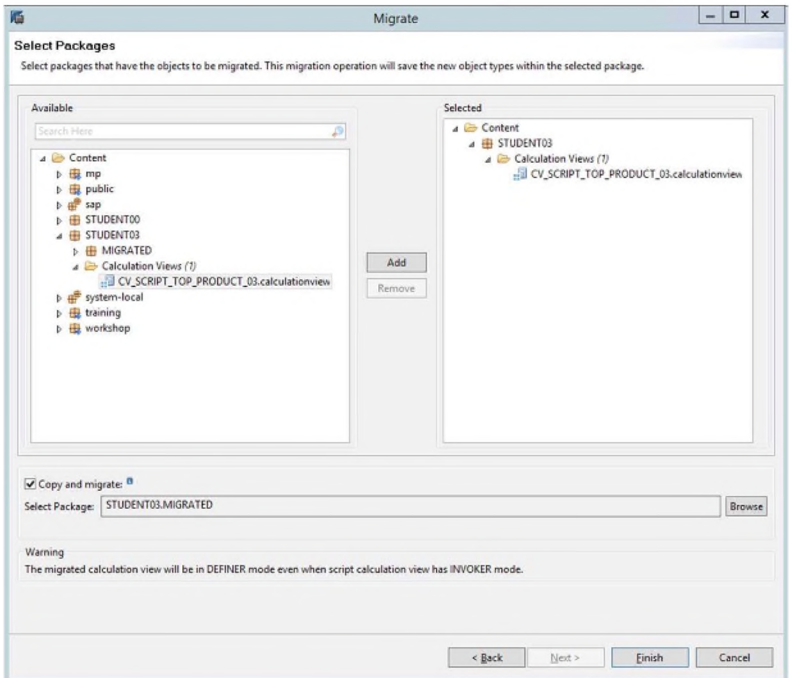


Figure 366: Selecting View and Defining Copy Options

6. Activate the generated Table Function.

> Hint:
> To see the Table Function in your *MIGRATED* package, you must select the *Show All Objects* checkbox in the preferences of SAP HANA Studio. This option is located in *SAP HANA → Modeler → Content Presentation*.

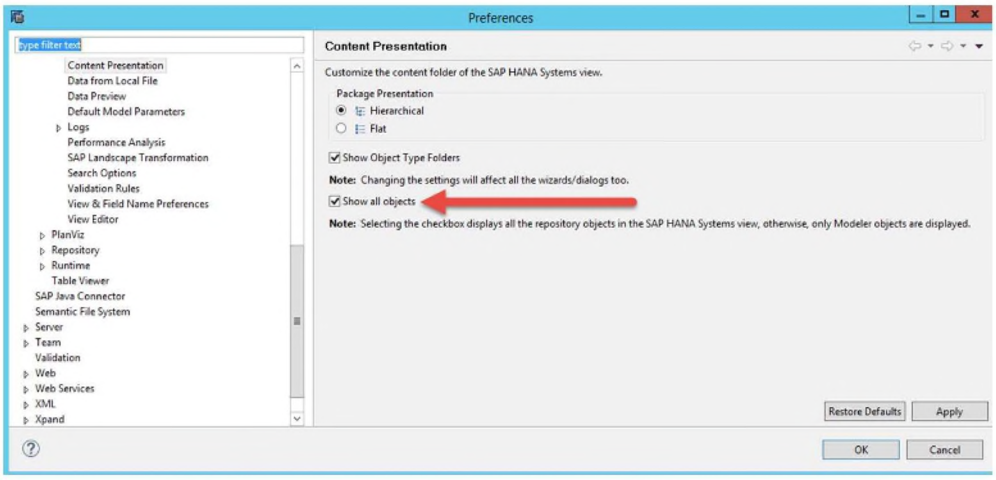


Figure 367: Show All Objects

a) In the *Systems* View, expand the *Content* folder to display your *STUDENT## → MIGRATED* package.

b) In the *Table Functions* folder, right-click the *TABLE_FUNCTION_CV_SCRIPT_TOP_PRODUCT_##.hdbtablefunction* object and choose *Activate*.

c) In the *Activate* window, confirm the selected object, keep the default activation mode, and click *Activate*.

d) In the *Job Logs* view, check that the status of the activation job is *Completed successfully*.

e) In the SAP HANA Studio preferences, deselect the *Show all objects* checkbox to come back to the default content presentation.

7. Visualize the new graphical Calculation View in your *STUDENT## → MIGRATED* package and note that the generated Table Function is used as a data source for the graphical Calculation View.

   a) In the *Systems* view, expand the *Content* folder to display the package *STUDENT## → MIGRATED → Calculation Views*.

   b) Double-click the view *CV_SCRIPT_TOP_PRODUCT_##*.

   c) In the *Scenario* pane, expand the *Aggregation* node and check its data source. The data source is the new table function *TABLE_FUNCTION_CV_SCRIPT_TOP_PRODUCT_##*.

8. Activate the new Calculation View

   a) In the *STUDENT## → MIGRATED* package.

   b) Right-click the view *CV_SCRIPT_TOP_PRODUCT_##* and choose *Activate*.

9. Perform a Data Preview from the generated graphical Calculation View in your *STUDENT##.MIGRATED* package.

   Follow the same steps as before, when previewing the data of for the scripted Calculation View, and ensure you achieve the same result in the *Raw Data* tab of the Data Preview.

   a) In the top-right of the *View* tab, choose *Data Preview*.

   b) Enter **5** as the input parameter for the number of years and choose *OK*.

   c) Select the *Raw Data* tab.

   d) Compare the results with the data preview of the original Calculation View.

LESSON SUMMARY

You should now be able to:

- Convert Attribute and Analytic Views into Calculation Views
- Convert Scripted Calculation Views into Table Functions

Learning Assessment

1. Which of the following documents displays all the details about an Information Object?

   Choose the correct answer.

   - [ ] A Model List
   - [ ] B Target List
   - [ ] C Model Detail
   - [ ] D Add List

2. Which of the following functionalities helps to identify the dependency of Information Models?

   Choose the correct answer.

   - [ ] A Validate
   - [ ] B Redeploy
   - [ ] C Where-Used
   - [ ] D Refactor

3. Identify the process involved in importing the models from your local system.

   Choose the correct answer.

   - [ ] A Create the delivery unit > Export the models to client > Import the models from client
   - [ ] B Create the delivery unit > Save the models > Import the models from client
   - [ ] C Define the delivery unit > Import the models from client > Save the models
   - [ ] D Export the models to client > Create the delivery unit > Import the models from client

4. A Delivery Unit is a group of transportable objects used for content delivery.

   Determine whether this statement is true or false.

   - [ ] True
   - [ ] False

5. You must migrate a set of information views in a single operation.

   Determine whether this statement is true or false.

   - [ ] True
   - [ ] False

6. What do you need to do after migrating an Attribute View before you can activate the new Dimension Calculation View?

   Choose the correct answer.

   - [ ] A Migrate all the other Attribute Views
   - [ ] B Migrate all the Analytic Views that reference this Attribute View
   - [ ] C Delete the source Attribute View

Learning Assessment - Answers

1. Which of the following documents displays all the details about an Information Object?

   Choose the correct answer.

   - [ ] A Model List
   - [ ] B Target List
   - [x] C Model Detail
   - [ ] D Add List

2. Which of the following functionalities helps to identify the dependency of Information Models?

   Choose the correct answer.

   - [ ] A Validate
   - [ ] B Redeploy
   - [x] C Where-Used
   - [ ] D Refactor

3. Identify the process involved in importing the models from your local system.

   Choose the correct answer.

   - [x] A Create the delivery unit > Export the models to client > Import the models from client
   - [ ] B Create the delivery unit > Save the models > Import the models from client
   - [ ] C Define the delivery unit > Import the models from client > Save the models
   - [ ] D Export the models to client > Create the delivery unit > Import the models from client

4. A Delivery Unit is a group of transportable objects used for content delivery.

   Determine whether this statement is true or false.

   [X] True

   [ ] False

5. You must migrate a set of information views in a single operation.

   Determine whether this statement is true or false.

   [ ] True

   [X] False

6. What do you need to do after migrating an Attribute View before you can activate the new Dimension Calculation View?

   Choose the correct answer.

   [ ] A Migrate all the other Attribute Views

   [X] B Migrate all the Analytic Views that reference this Attribute View

   [ ] C Delete the source Attribute View

UNIT 7 Security in SAP HANA Modeling

UNIT OBJECTIVES

- Understand roles and privileges
- Explain how Users, Roles and Privileges are used to implement Objects and Data Access security in SAP HANA
- Define privileges
- Explain how to grant the relevant privileges to the Modeler and End-User
- Generate SAP HANA Live Privileges
- Convert Classical Analytic Privileges to SQL-Based Analytic Privileges

Unit 7
Lesson 1

Understanding Roles and Privileges

LESSON OVERVIEW

This lesson will give you an overview of how security is implemented in SAP HANA.

Business Example

After the SAP HANA System has been installed, you need to define security and give the relevant authorizations to users so that they can administrate the system, provision data, and, in the case of the Modeler role, start creating models.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Understand roles and privileges
- Explain how Users, Roles and Privileges are used to implement Objects and Data Access security in SAP HANA

Overview of User Management and Security

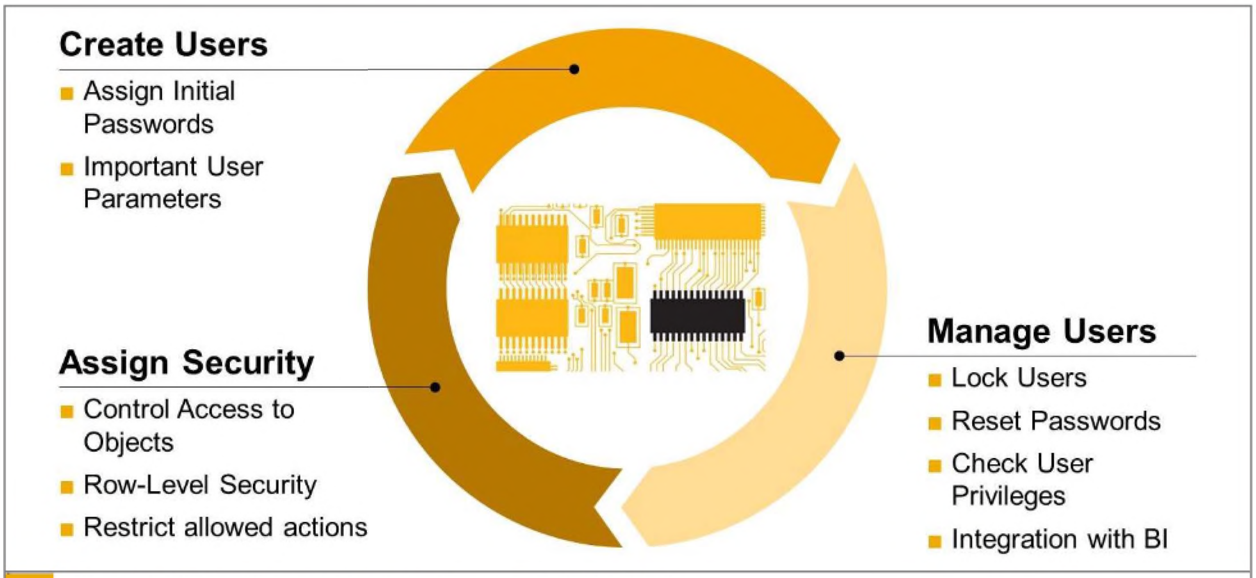


Figure 368: User Management and Security in SAP HANA

Like any other database system, SAP HANA provides features to define and maintain security.

The reasons for implementing security are as follows:

- Restrict database administration to skilled and empowered employees only
- Ensure that each database user has relevant authorization to manipulate database objects
- Separate duties

- Manage access to the system from a variety of front-end tools
- Restrict access to the data stored in the system based on role, geographic, or organizational responsibilities

Users, Roles and Privileges

- A known user can log on to the database. A user can be the owner of database objects.
- A role is a collection of privileges and can be granted to either a user or another role (nesting).
- A privilege is used to enable a specific operation on one or several specific objects.

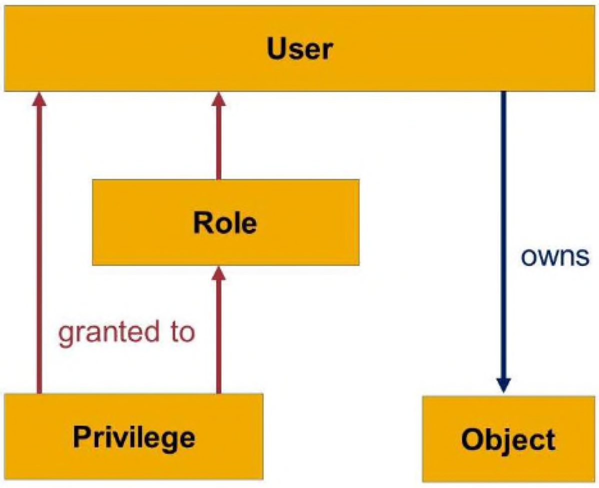


Figure 369: Users, Roles, Objects and Privileges

The figure, Users, Roles, Objects and Privileges, shows how Users and Roles can be granted privileges.

- Privileges can be assigned to users either directly, or indirectly by using roles.

  Roles help you to structure the access control scheme and model reusable business roles. They can be nested, enabling the implementation of a hierarchy of roles.

> Hint:
> It is highly recommended to manage authorizations for users by using roles. Assigning a privilege directly to a user is not a good practice.

- All the privileges granted directly or indirectly to a user are combined.

  Whenever a user tries to access an object, the system performs an authorization check based on the user's roles and directly allocated privileges (if any).

- It is not possible to explicitly deny privileges.

  In particular, the system does not need to check all the users roles. As soon as all the privileges required for a specific operation on a specific object have been found, the system ends the check and allows the operation.

- Several predefined roles exist in the SAP HANA database.

  Some of them are templates (and need to be customized), and others can be used as they are.

- As a best practice, users should only be given the smallest set of privileges required for their role.

Defining a Hierarchy of Roles and Privileges

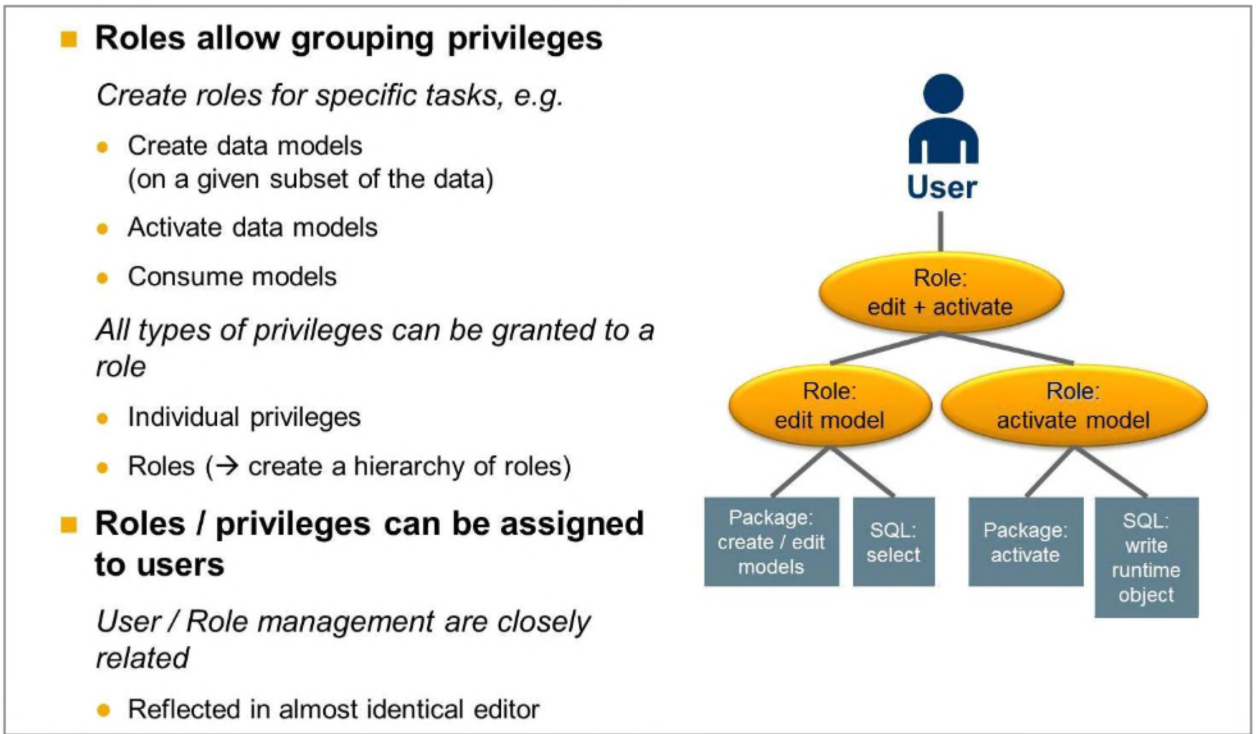


Figure 370: Defining a Hierarchy of Roles and Privileges

Defining Roles

There are two ways to create roles:

- As pure runtime objects that follow classic SQL principles (Catalog Roles)
- As design-time objects in the repository of the SAP HANA database (Repository Roles)

Table 22: Catalog Roles vs. Repository Roles

Feature	Catalog Roles	Repository Roles
Transportability	No	Yes
Version Management	No	Yes
Relationship to database users	Each role is owned by the database user that created it.	All roles are owned by the technical user *_SYS_REPO*.
Grant and revoke process	Roles are granted by the database user with the SQL *GRANT* statement.	Roles are granted and revoked using built-in procedures. An *EXECUTE* privilege on these procedures is the only authorization needed to grant or revoke any role.

In general, repository roles are recommended, as they offer more flexibility.

In particular, they can easily be transported between SAP HANA Systems. For example, application developers can model role templates and ship them together with the application.

> Note:
> Repository Roles are created as *.hdbrole* files within the *SAP HANA Developer* perspective. They cannot be created from the *Modeler* perspective.

On the other hand, a typical use case where Catalog Roles can be used is when security is managed in another tool, and the SAP HANA database is only accessed by a very limited number of technical users.

How to Assign Privileges

The ways to maintain users and roles in SAP HANA are as follows:

- In SAP HANA studio, by using the dedicated folder *Security* of the *Systems* view.
- From the SQL Console, by executing SQL statements.
- By executing one of the *GRANT_...* or *REVOKE_...* built-in procedures in the catalog of user *_SYS_REPO*.

  This option is the one that provides most flexibility for Repository Roles, because the roles are granted by the technical user *_SYS_REPO*, so they are never revoked, even if the actual granting user is deleted from the system.

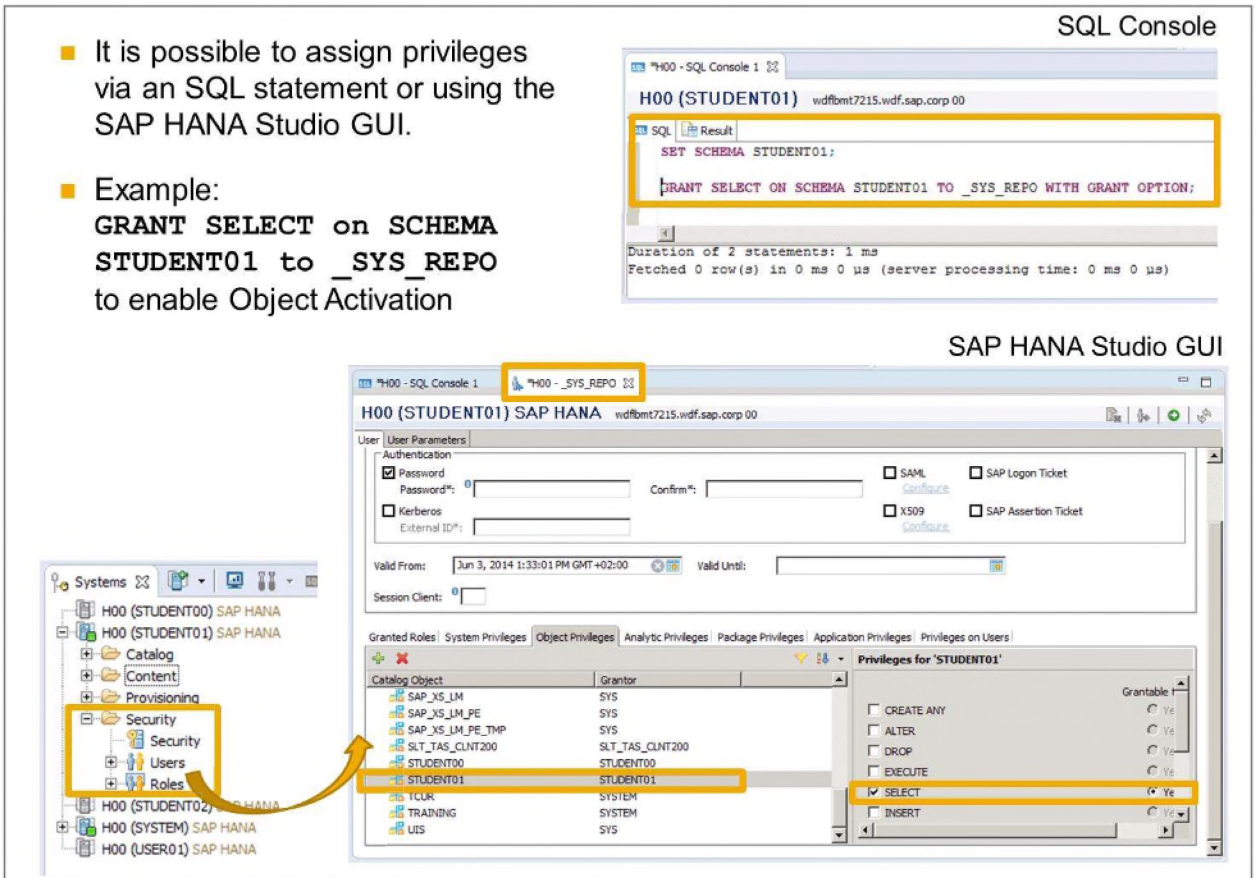


Figure 371: Assigning Privileges by SQL or HANA Studio UI

The figure, Assigning Privileges by SQL or HANA Studio UI, shows how to grant *SELECT* privileges on schema *STUDENT01* to the technical user *_SYS_REPO*, with the *Grant* option.

Managing Users and Roles

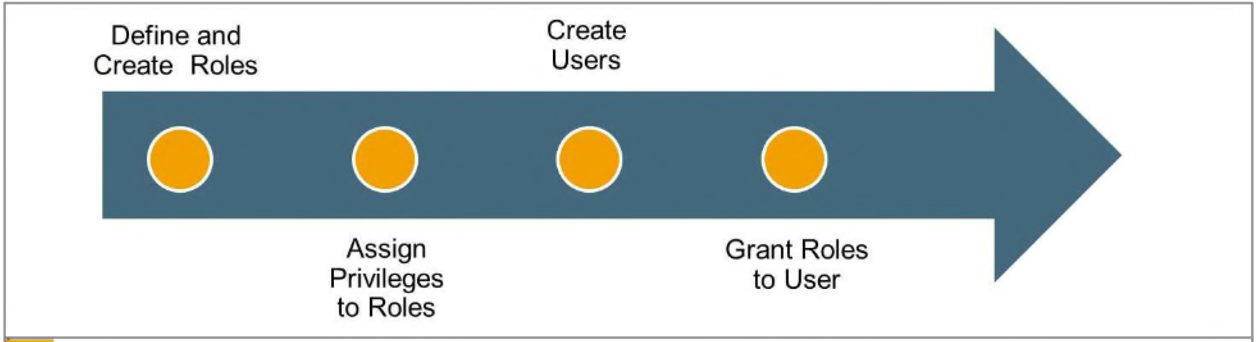


Figure 372: Managing Users and Roles

The figure, Managing Users and Roles, is the general process to define users and roles and assign privileges.

You can now have a quick hands-on to get familiar with the user interface that is used to assign roles to users. You will then get into more detail about the different types of privileges that can be assigned to roles (and also to users, but as already said this should be avoided in most circumstances).

Related Information

You can find more information about security in the following guides, available on the SAP Help Portal. Go to http://help.sap.com/hana):

- SAP HANA Security Guide
- SAP HANA Modeling Guide for SAP HANA Studio

You can also refer to the training course HA240 - SAP HANA Security & Authorization. Go to http://training.sap.com for more details.

Unit 7
Exercise 30

Work with SAP HANA Users and Roles

Exercise Objectives

After completing this exercise, you will be able to:

- Review SAP HANA users and roles

Business Example

As a consultant in charge of modeling, you have created an analytic view for your customer. The customer now tries to open the view, but gets a *Missing Authorization* error.

You must analyze and solve this issue.

> Note:
> To simulate this situation, an additional user, with fewer privileges than your *STUDENT##* user, is available in the SAP HANA database. The credentials of this user are as follows:
>
> - User name: **`USER##`**
> - Password: **`Training1`**
>
> The two letters ## represent the two-digit group number assigned to you by the instructor.

Task 1: Review the Privileges Assigned to USER## and try to Preview Data with this User.

> Note:
> Before this exercise can begin, the instructor needs to change temporarily setting to allow SAP HANA to capture authorization problems in the database trace file. Please check that the instructor has completed this task before.

1. Review the granted roles for the additional user, *USER##* (## is your group number).

   Which roles are granted to *USER##*?

   ___

   ___

   ___

2. Connect to the same SAP HANA system *H00* with a different user: *USER##*.

   The password for this user is **`Training1`**.

> Hint:
> Right-click the *H00 (STUDENT##)* entry in the *Systems* view.

3. Connect to the SAP HANA System with *USER##*, try to preview the data of the Calculation View *CVCS_PO2*, located in the *TRAINING* → *ha300* package. Choose the *Raw Data* tab. What is the result?

> Hint:
> To connect to the SAP HANA System with *USER##*, expand the root node *H00 (USER##) SAP HANA* in the *Systems* view and open the Analytic View from the corresponding *Content* folder.

Task 2: Analyze the Root Cause for the Authorization Issue

You will now review the relevant trace file to identify the exact reason why *USER##* cannot view the data of the Calculation View.

1. Open the most recent trace file *indexserver_wdflbmt7215*...*.trc*. This file can be found in the *Diagnosis Files* tab of the *Administration* view

> Hint:
> To display the *Administration* view, double-click the root node *H00 (USER##)* in the *Systems View*.

- Select the header of the *Modified* column to sort the files by modification time.
- You can filter the *Diagnosis Files* list by entering **`index*trc`** in the *Filter:* field.

2. Search the trace file for details about the authorization issue for user *USER##*.

> Hint:
> Search the file for the character string **`USER##`**.

Unit 7
Solution 30

Work with SAP HANA Users and Roles

Exercise Objectives

After completing this exercise, you will be able to:

- Review SAP HANA users and roles

Business Example

As a consultant in charge of modeling, you have created an analytic view for your customer. The customer now tries to open the view, but gets a *Missing Authorization* error.

You must analyze and solve this issue.

> Note:
> To simulate this situation, an additional user, with fewer privileges than your *STUDENT##* user, is available in the SAP HANA database. The credentials of this user are as follows:
>
> - User name: **`USER##`**
> - Password: **`Training1`**
>
> The two letters ## represent the two-digit group number assigned to you by the instructor.

Task 1: Review the Privileges Assigned to USER## and try to Preview Data with this User.

> Note:
> Before this exercise can begin, the instructor needs to change temporarily setting to allow SAP HANA to capture authorization problems in the database trace file. Please check that the instructor has completed this task before.

1. Review the granted roles for the additional user, *USER##* (## is your group number).

   Which roles are granted to *USER##*?

   [Correct answer]

   a) In the *Systems* view, locate the folder *Security* → *Users* under your *STUDENT##* connection to the system.

   b) Expand the *Users* folder. Find user *USER##* (## is your group number).

c) Right-click *USER##* and choose *Open*.

d) Review the *Granted Roles* tab and check that it contains the following roles:

- PUBLIC
- RESTRICTED_ROLE

2. Connect to the same SAP HANA system *H00* with a different user: *USER##*.
   The password for this user is **`Training1`**.

> Hint:
> Right-click the *H00 (STUDENT##)* entry in the *Systems* view.

a) In the *Systems* view, right-click the existing system connection and choose *Add System with Different User*.

b) Enter the user name and password.

c) Choose *Finish*.

3. Connect to the SAP HANA System with *USER##*, try to preview the data of the Calculation View *CVCS_PO2*, located in the *TRAINING → ha300* package. Choose the *Raw Data* tab. What is the result?

> Hint:
> To connect to the SAP HANA System with *USER##*, expand the root node *H00 (USER##) SAP HANA* in the *Systems* view and open the Analytic View from the corresponding *Content* folder.

a) In the *Systems* view, in the package *TRAINING → ha300*, right-click the Calculation View *CVCS_PO2* and choose *Data Preview*.

b) Select the *Raw Data* tab.
   The results cannot be displayed. You receive a *"insufficient privilege"* error message.

Task 2: Analyze the Root Cause for the Authorization Issue

You will now review the relevant trace file to identify the exact reason why *USER##* cannot view the data of the Calculation View.

1. Open the most recent trace file *indexserver_wdflbmt7215*...*.trc*. This file can be found in the *Diagnosis Files* tab of the *Administration* view

> Hint:
> To display the *Administration* view, double-click the root node *H00 (USER##)* in the *Systems View*.

- Select the header of the *Modified* column to sort the files by modification time.
- You can filter the *Diagnosis Files* list by entering **`index*trc`** in the *Filter:* field.

a) In the *Systems* view, double-click the system connection with user *USER##*.

b) Select the *Diagnosis Files* tab.

c) Find and double-click the most recently modified file with name *indexserver_wdflbmt7215*...*.trc.* .

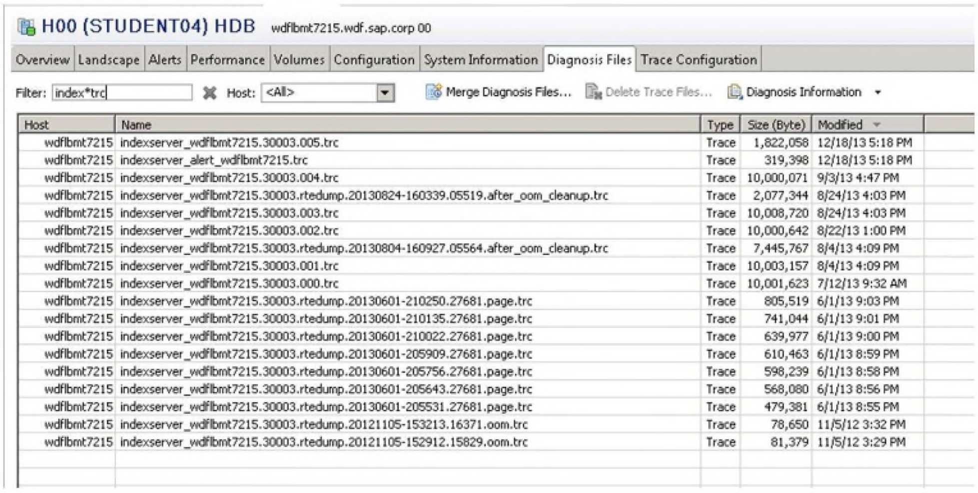


Figure 373: Diagnosis Files

2. Search the trace file for details about the authorization issue for user *USER##*.

> Hint:
> Search the file for the character string **USER##**.

a) Press Ctrl+F, enter **USER##** and choose *Find*.

b) Review the missing authorization messages.

The trace log indicates an error on missing *Analytical Privileges*:

```
[8853]{215311}[234/-1]
2015-12-18 17:29:32.776881 i TraceContext
        TraceContext.cpp(00699) : UserName=USER04,
...
[8853]{215311}[234/-1] 2015-12-18 17:29:32.776870 i Authorization
        XmlAnalyticalPrivilegeFacade.cpp(01252) : UserId(143277)
        is missing analytic privileges in order to access
...
Current situation: AP ObjectId(13,2,oid=3): Not granted.
 ...
```

As a conclusion, USER## requires a Analytic Privilege to be able to preview the data of the Analytic View.

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

LESSON SUMMARY

You should now be able to:

- Understand roles and privileges
- Explain how Users, Roles and Privileges are used to implement Objects and Data Access security in SAP HANA

Unit 7
Lesson 2

Defining Privileges

LESSON OVERVIEW

This lesson will give you an introduction to the different types of privileges that are used to define authorizations in an SAP HANA system.

The main focus will be on the roles of the Modeler, who creates, tests, and manages information models, as well as the End-User, who consumes the data using information models.

Business Example

The SAP HANA Security Model contains several types of Privileges.

You have been asked to review the privileges and make sure they correspond to the defined Security Model.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Define privileges
- Explain how to grant the relevant privileges to the Modeler and End-User

Types of Privileges

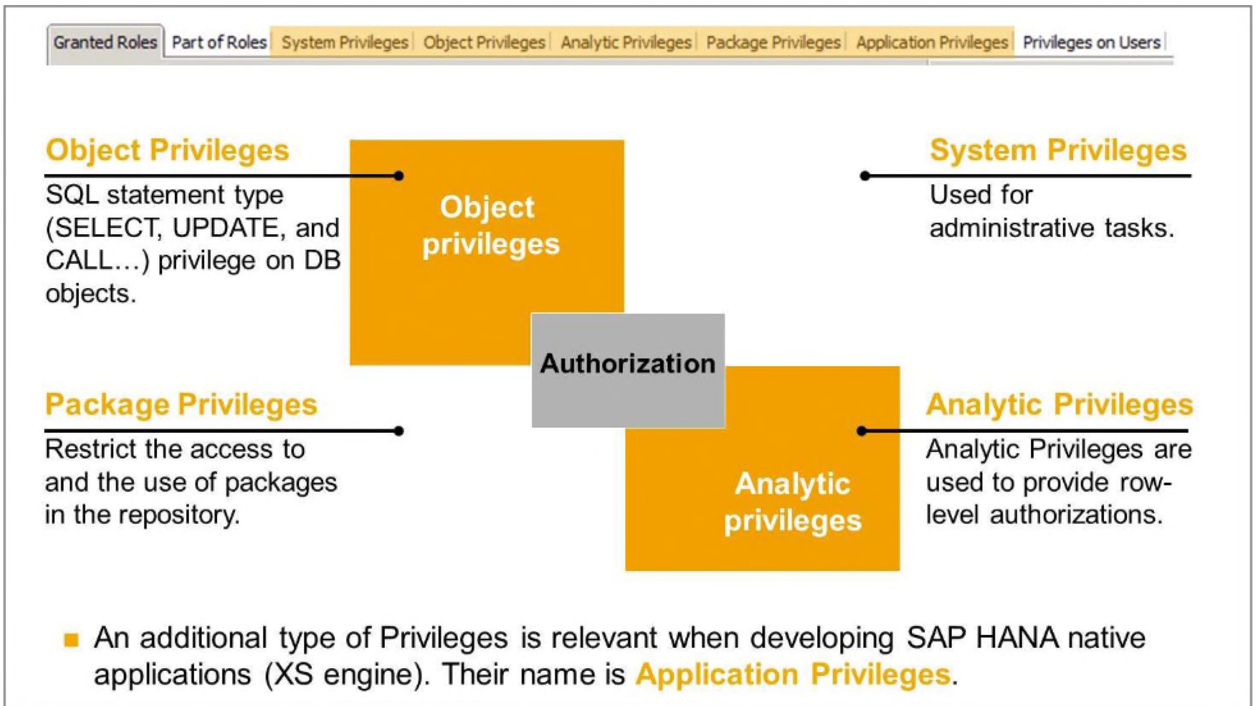


Figure 374: Authorization Concept - Types of Privileges

Package Privileges and Analytic Privileges are the most important for the Modeler and End-User roles. Authorization in SAP HANA builds on these types of privileges, such as the following:

System Privileges

System Privileges are used to control general system activities.

They include general administrative actions, such as creating or deleting schemas, managing users and roles, performing backup, monitoring and tracing, and so on.

System Privileges also include several privileges directly related to the modeling activities, such as maintaining, exporting, and importing delivery units.

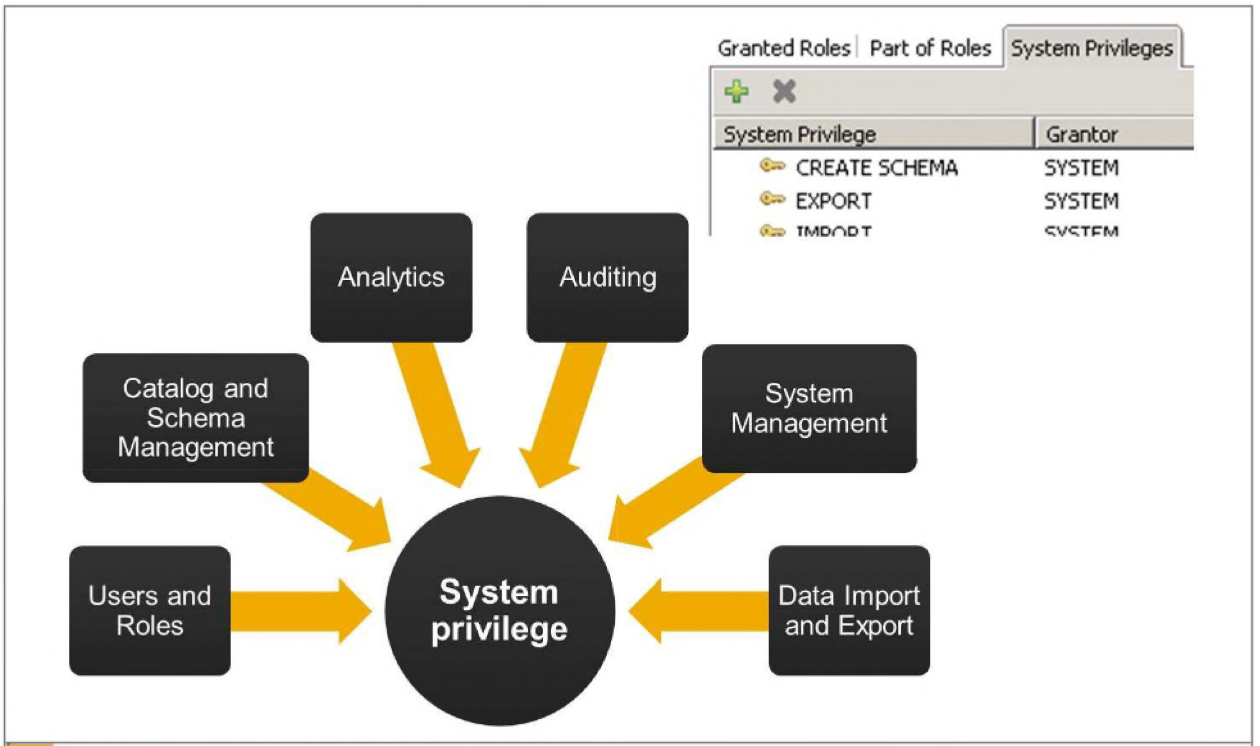


Figure 375: System Privileges

Object Privileges

Object Privileges are used to allow access to and modification of database objects.

Each Object Privilege is related to one object and, depending of the type of object (schema, table, and procedure) includes the different SQL statement types, which you can grant separately; for example, `CREATE`, `ALTER`, `DROP`, `SELECT`, `INSERT`, `UPDATE`, `DELETE`, and `EXECUTE`.

> Note:
> For most of these actions, whether you allow the user role that has the right to perform the action to grant it to other users.

- Object privileges are bound to an object, for example, to a database table or schema, and enable object-specific control activities, such as SELECT, UPDATE, or DELETE.

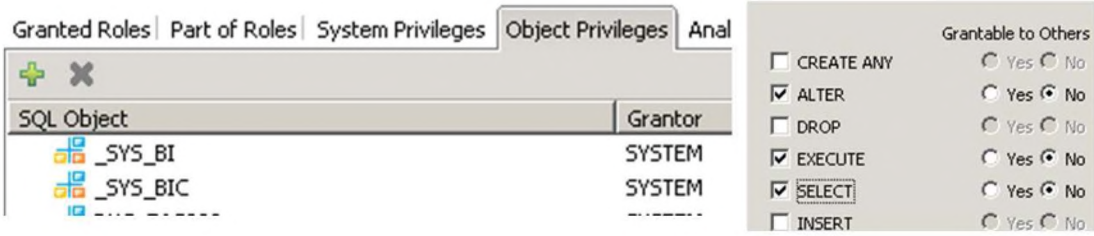


Figure 376: Object Privilege

Package Privileges

Package Privileges are used to allow the access to and the modification or execution of information models from Packages in the repository of the SAP HANA database.

Packages contain design-time versions of various objects, such as Analytic, Attribute, and Calculation Views, as well as Analytic Privileges and Procedures. To be able to work in packages, you must be granted the corresponding Package Privileges.

For each package, the privilege can include the following actions: Read, Edit, Activate, and Maintain. You have to specify whether the right to perform the action can be granted to other users.

- In the SAP HANA studio, you can manage the package privileges on the *Package Privileges* tab.
- The SAP HANA database repository is structured hierarchically with packages assigned to other packages as sub-packages. If you grant privileges to a user for a package, the user is automatically also authorized for all corresponding sub-packages.

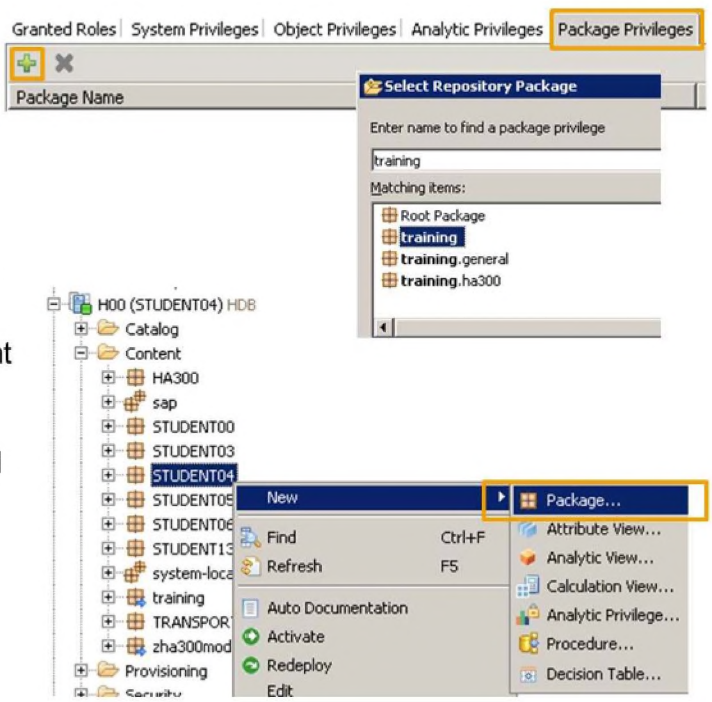


Figure 377: Package Privileges

Analytic Privileges

Analytic Privileges are used to enable data access to different portions of data in Analytic, Attribute, and Calculation Views, by filtering the attribute values.

Each information view has a dedicated property indicating whether the execution of the view must be restricted by Analytic Privileges or not. If you choose to restrict the view's data, you

have to opt for Classic or SQL Analytic Privileges. You will learn more about this distinction later.

Analytic Privileges must be defined and activated before they can be granted to roles (or users).

> Note:
> An Analytic Privilege is evaluated when a user executes a query against one of the information views to which the Privilege applies.

If we consider the SAP HANA Native Application development, an important type of authorization is the Application Privileges.

Application Privileges

Developers of SAP HANA XS applications can create application privileges to authorize user and client access to their application. They apply in addition to other privileges.

It is recommended to grant application privileges to roles created in the SAP HANA Repository at design time. The distinction between Catalog roles and Repository roles is important.

> Note:
> An additional type of privilege, *Privilege on User*, exists in the definition of users and roles.
>
> It is used for debugging purposes. Each user can grant the right *ATTACH DEBUGGER* to another user to allow them to help debug an external session (that is, the session of the user who had granted this right).

System Privileges Applicable to the Modeler Role

Several System Privileges are relevant to the Modeler role.

- As part of the System Privileges, the following authorizations are relevant for the modeler role:

Privilege	Description
REPO.EXPORT	Export Delivery Units
REPO.IMPORT	Import Delivery Units
REPO.MAINTAIN_DELIVERY_UNITS	Create, Modify and Delete Delivery Units
REPO.WORK_IN_FOREIGN_WORKSPACE	Work in a foreign inactive workspace. In particular, this role is needed to take over an inactive object located in the repository of another user.

Figure 378: System Privileges Relevant for the Modeler Role

In addition, the following roles are specific to the use of SAP HANA Change Recording, which is part of SAP HANA Application Lifecycle Management:

- REPO.CONFIGURE
- REPO.MODIFY_CHANGE
- REPO.MODIFY_OWN_CONTRIBUTION
- REPO.MODIFY_FOREIGN_CONTRIBUTION

Package Privileges

Package Privileges are used to authorize actions on individual packages of the SAP HANA Repository.

Package Privileges are cascaded from one package to all its sub-packages. You can grant additional privileges to a sub-package, but it is not possible to revoke a privilege in a sub-package that comes from the parent package.

- On each package, you can grant the following privileges:

Privilege	Description
REPO.READ	List the content of packages
REPO.EDIT_**_OBJECTS	Edit Information Views
REPO.ACTIVATE_**_OBJECTS	Activate Information Views
REPO.MAINTAIN_**_PACKAGES	Update or Delete packages, create sub-packages

** stands for "NATIVE" or "IMPORTED"
Authorizations to Edit or Activate Objects and to Maintain Packages can be defined separately for NATIVE and IMPORTED packages

- Privileges explicitly granted to a package are automatically granted to its sub-packages.

  You can grant additional privileges to a sub-packages, but you cannot revoke a privilege that is implicitly passed from a parent package.

- You can grant privileges to the Root package.

Figure 379: Package Privileges

Native and Imported Package Privileges

Apart from the REPO.READ privilege, the other package privilege can be applied differently on Native and Imported packages.

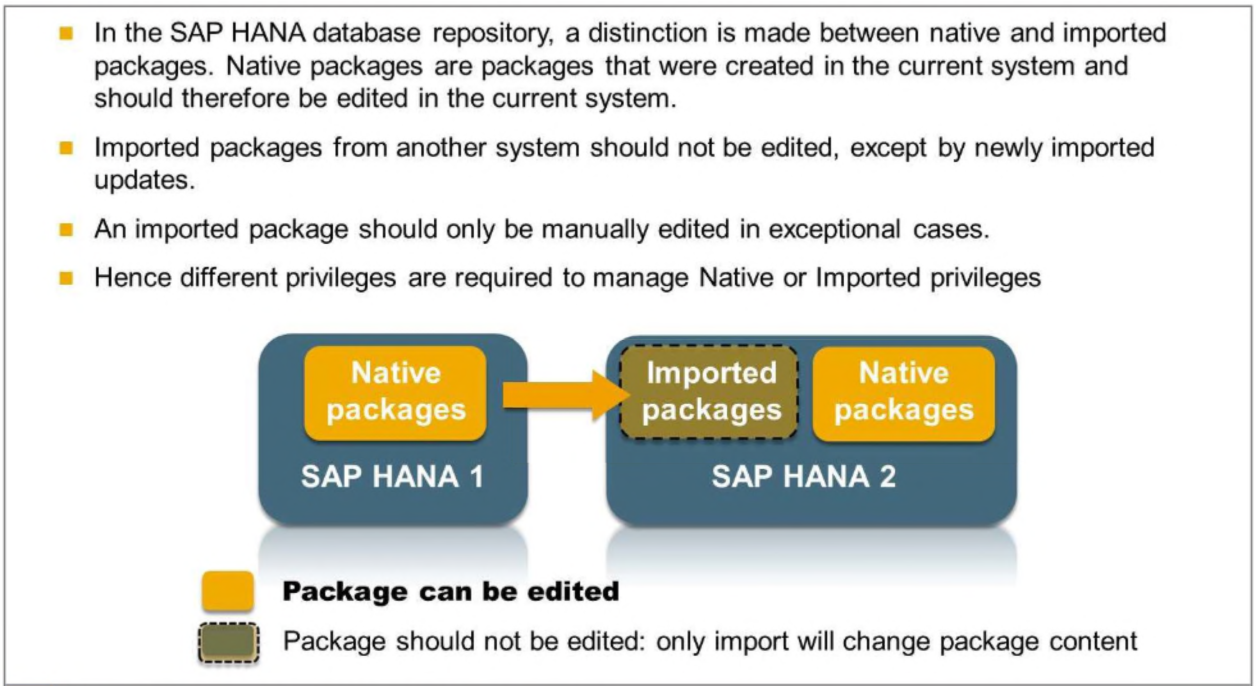


Figure 380: Native and Imported Package Privileges

Package privileges are cascaded and have distinct authorizations for native and imported packages. This makes it possible to protect the content of imported packages.

For example, if you grant a user the *REPO.EDIT_NATIVE_PACKAGES* (but not the *REPO.EDIT_IMPORTED_PACKAGES*) on the *Root Package*, this user will not be able to modify an imported package, but will be able to modify any native package.

> Note:
> The same example applies to any package or sub-package. Depending on your business requirement, system complexity, and security implementation approach, you would not necessarily grant package privileges at the upmost level (the *Root Package*).

Object Privileges Applicable to the Modeler or End-User Roles

The consumption of information views in SAP HANA requires a number of Object Privileges granted to the modeler and the end-user, such as the following:

- *EXECUTE* privilege on procedure *REPOSITORY_REST (SYS)* (Modeler Role).

  The modeler needs this privilege to access the repository in SAP HANA studio.

- *SELECT* privilege on the tables referenced by the information view (Modeler Role).

  The modeler needs this privilege to create, modify, and activate the view.

  > Note:
  > This privilege is NOT needed to query the information view.

- *SELECT* privilege on the *_SYS_BI* schema. (Modeler and End-User Role).

  This schema contains several tables *BIMC_xxx* that define the list of all information views and their properties, such as, columns, and variables.

- *SELECT* privilege on the activated version of the information view, located in the schema *_SYS_BIC*.

  The activated version of the information view appears in the object list as *<package>.[<subpackage>]/<viewname> (_SYS_BIC)*. For example, *STUDENT00.ha300/CVCS_BP2_00 (_SYS_BIC)*.

> Note:
> If an information view, V1, references another information view, V2, querying the view V1 only requires SELECT privilege on view V1. No select privilege on view V2 is needed.

Depending on the client tools, some additional privileges need to be granted to the end-user.

For example, querying an information view with MS Excel (MDX) creates temporary tables in schema _SYS_BIC, and requires that the end-user have more authorizations than only *SELECT* on the *_SYS_BIC* schema.

Analytic Privileges

Analytic Privileges are used to control access to SAP HANA data models

- Can be Applied to the following objects:
  - Calculation Views
  - Attribute Views (deprecated)
  - Analytic Views (deprecated)

You use Analytic Privileges to implement row-level security

- Restrict access to a given data container to selected Attribute Values:
  - Columns from *DIMENSION* Calculation View
  - Shared or Private Attribute Columns in *CUBE* Calculation Views
  - Columns in Attribute Views (deprecated)
  - Shared or Private Attribute Columns in Analytic Views (deprecated)
  - Combinations of the above
- Restrictions support *Single value*, *Range*, and *In list*

Figure 381: Analytic Privileges

Analytic Privileges are used to enable data access to different portions of data in Information Views, by filtering the data based on the values of one or several attributes.

The rationale for Analytic Privileges is to allow the use of Information Views by different users who might not be allowed to see the same data.

For example, different regional sales managers who are only allowed to see sales data for their regions could use the same Information View, but you would have to define and assign Analytic Privileges to the view so that each manager sees only the data for their region.

Note:
This is different from the behavior of SAP Business Warehouse (BW). While the concept is similar, SAP BW will forward an error message if you execute a query that returns values you are not authorized to see.

With SAP HANA, the query would be executed and, depending on your authorizations, only values that you can see would be returned.

Types of Analytic Privileges

There are two different types of Analytic Privileges

Two Flavors of Analytic Privileges

- Classical Analytic Privileges (also called XML-based Analytic Privileges)

  This is the historical Data Access restriction type, which now tends to be deprecated and replaced with the SQL–based one.

- SQL Analytic Privileges

  This new type of Analytic Privilege has been introduced with SAP HANA SPS10. It provides more flexibility in scenarios where filter conditions are complex.

Caution:
In each Information View, you decide in the properties whether the access to the data of the view should be restricted by Analytic Privileges. If so, you explicitly decide which type of Analytic Privileges (Classic or SQL) the view will use.

For this reason, each type of Analytic Privileges can only secure Information Views that check this type of Analytic Privilege. For example, if the list of views secured by a SQL Analytic Privilege APS_1 contains views that mention Classical Analytic Privilege in their properties, the privilege APS_1 will not be checked.

The following sections will first focus on Classical Analytic Privileges, as most of the features of these privileges are available in SQL Analytic Privileges. Just after that, you will learn more about the specifics of SQL Analytic Privileges.

Analytic Privilege - Start Creation Wizard

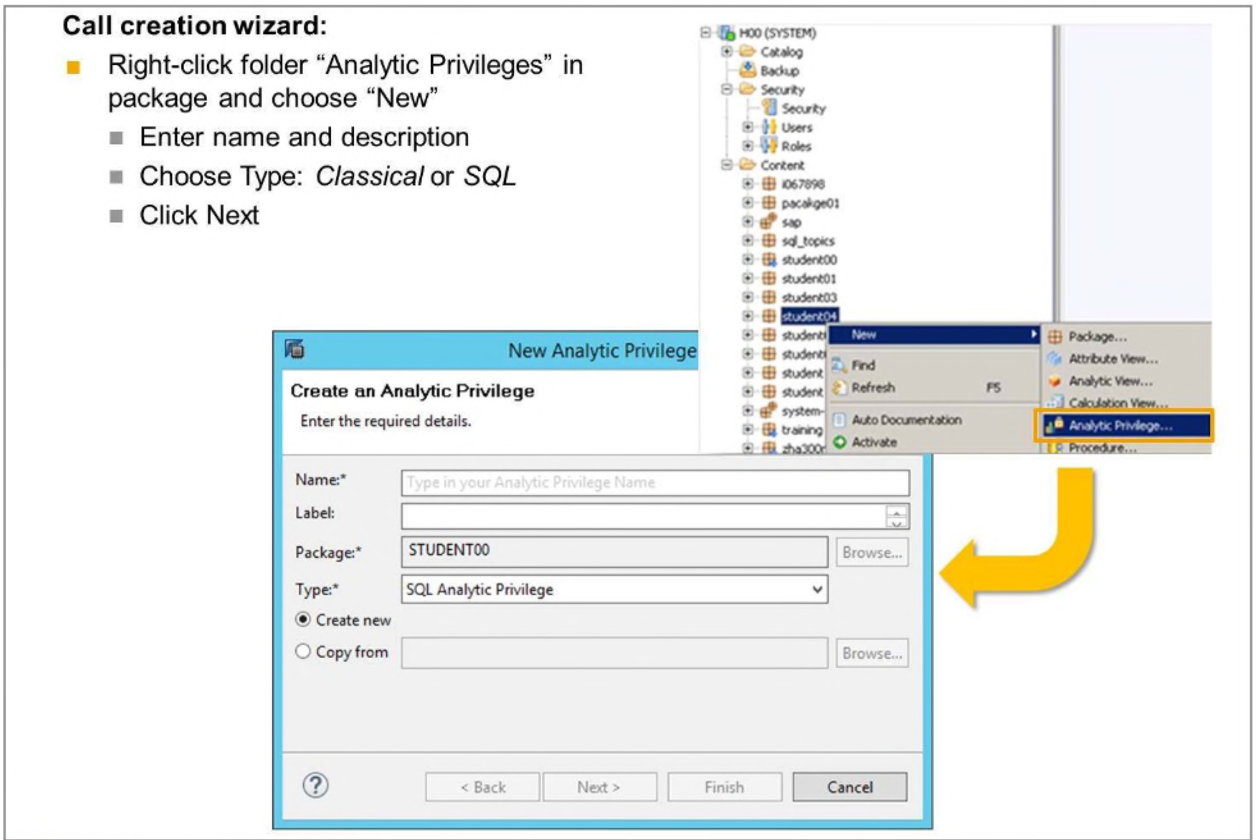


Figure 382: Analytic Privilege - Start Creation Wizard

Analytic Privileges of both types (Classic and SQL) are repository objects. You can create them in a different package than the one that contains the information views you want to secure.

After creating Analytic Privileges, they must be activated before they can be granted to roles (or users).

Analytic Privileges can be transported by using a Delivery Unit (especially along with the information views that they secure), which makes them a flexible tool to implement data access security.

Analytic Privilege - Select Information Models

To design an Analytic Privilege, you must consider the following properties:

- List of Information Views

  An Analytic Privilege defines the list of information views to which it is applicable.

  These views can be selected in any package, not only the package in which the Analytic Privilege is located.

You can add Information Views when you create the Analytic Privilege, or select them later.

Select applicable Information Models

- The views selected to define an Analytic Privilege determine:
  - Which views the Analytic Privilege will apply to
  - What attributes you can choose to define restrictions
- You can add other views to an existing Analytic Privilege at any time

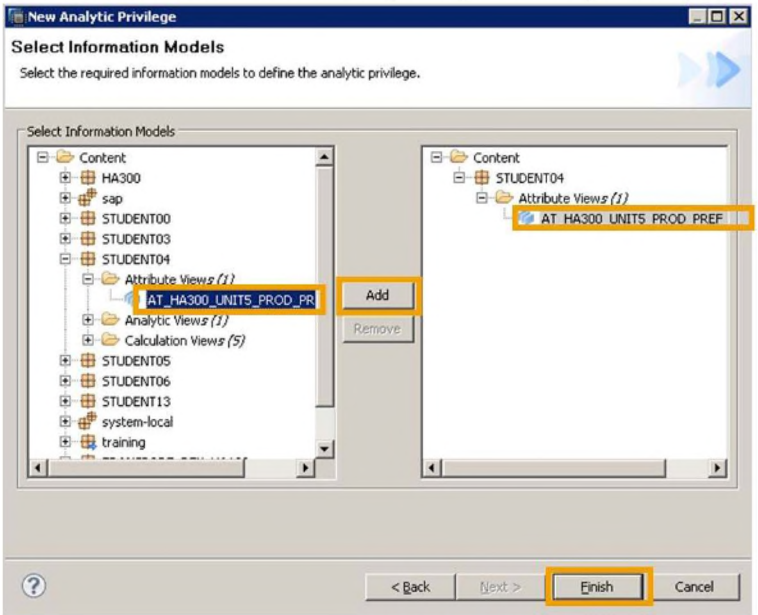


Figure 383: Analytic Privilege - Select Information Models

Analytic Privilege - Editor Overview

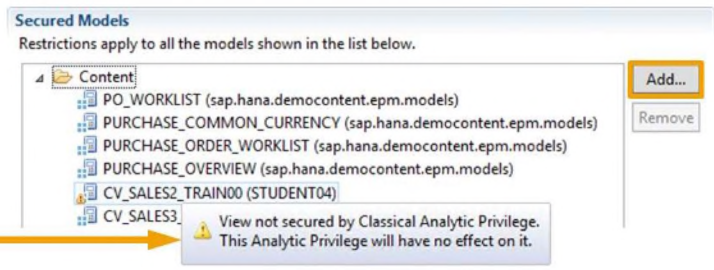
The Analytic Privilege editor lists, at any time, all the models that are secured by the Analytic Privilege.

It provides a useful indication on whether the Analytic Privilege will apply to each selected view, by displaying a warning icon and a contextual information on mouse hover, in the following cases:

- When a selected information view is designed to NOT check Analytic Privileges.
- When a selected information view is designed to check Analytic Privileges of the other type (Classic vs. SQL or vice versa).

To add a model to the *Secured Models* list

- Click *Add*, and choose
  - any appropriate view
  - from any package
- Warning provided if the model design not in sync with the type of Analytic Privilege



Do not use the *Applicable to all information models* option

- It can have surprising side-effects
- You give away control over data access



Figure 384: Analytic Privilege - Editor Overview

> Caution:
> It is recommended not to select *Applicable to all information models*.

- Validity

  The Validity of an Analytic Privilege defines the time period the Analytic Privilege is valid.

- Associated Attributes and Restrictions

  An Analytic Privilege defines the Attributes on which restrictions will apply.

  You can select one or several attribute columns from the views referenced by the Analytic Privilege.

  For each associated attribute, you define a restriction, which can contain multiple value filters. Each value filter is a tuple of an operator and its operands, which is used to represent the logical filter condition.

  For example, a value filter *Equals 2016* can be defined for a dimension attribute *YEAR* in a dimension restriction to filter the data to year 2016 only.

> Note:
> Only columns of type *Attribute* (not *Measure*) can be specified in dimension restrictions.

Analytic Privilege - Select Column for Attribute Restriction

You can implement value restrictions for all the selected attributes

- If no value restriction is assigned → no restriction (same behavior as if you do not select the attribute)
- Otherwise → user will only be allowed to see listed values

- UI offers *single value* and *range* condition
- Several conditions on the same attribute are combined with a logical "OR"
- Several conditions on different attributes are combined with a logical "AND"

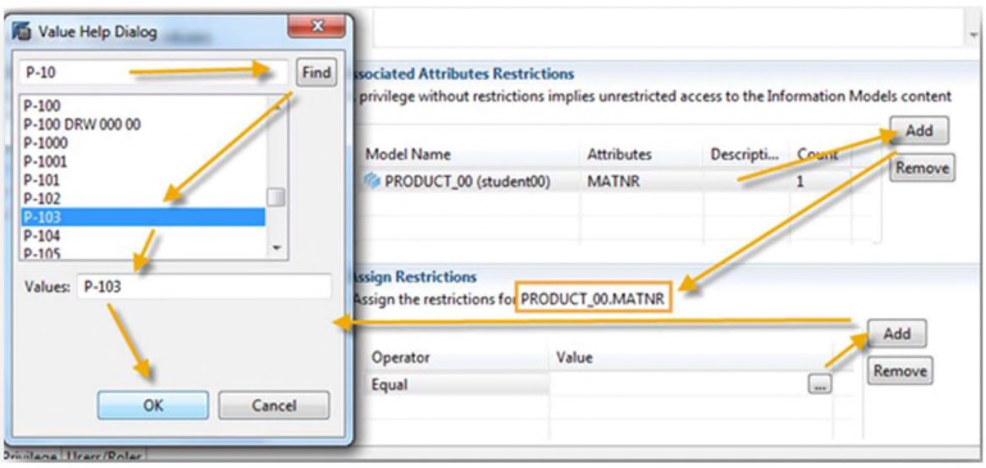


Figure 385: Analytic Privilege - Select Column for Attribute Restriction

Several restrictions filters within an Analytic Privilege are combined in the following way:

- Within one attribute column, all value filters are combined with a logical `OR`.
- Within one Analytic Privilege, all Dimension restrictions are combined with a logical `AND`.

For example, if an Analytic Privilege includes two restrictions on two different attributes (YEAR=2008, COUNTRY=US), the user is allowed to see only data fulfilling the compound condition `YEAR=2008 AND COUNTRY=US`.

Caution:
If two Analytic Privileges (or more) are defined to apply to the same view, SAP HANA combines the corresponding conditions with a logical “OR”.

If the two restrictions from the previous example were defined in two different Analytic Privileges applying to this user and this view, the user would see more data. Namely all the rows for which `YEAR=2008 OR COUNTRY=US` (that is, any year for COUNTRY=US and any country for YEAR=2008).

Assign Restrictions to Attributes

To assign restrictions to attributes, the main options include the following:

- Assign a *Fixed* restriction filter, applicable to all the users that will be granted the Analytic Privilege.

  For example, “Filter sales order for which the buyer is located *in the US*”
- Assign a restriction filter that is *based on a Catalog or Repository Procedure* that passes values dynamically, especially values that depend on the properties of the user.

  For example, “Filter sales order for which the buyer is located “in a country that is in the portfolio of the user”. In this case, the Procedure should retrieve the list of countries in the user's portfolio.

Note:
This type of privilege is called a *Dynamic Analytic Privilege*, and will be discussed later in this lesson.

Define Value in the Restriction Filter

To define value in the restriction filter, you can use the following operators:

- **Between <scalar_value_1> <scalar_value_2>**
- **ContainsPattern <pattern with *>**
- Comparison operators: **=, <=, <, >, >=** with **<scalar value>**
- **IsNull** and **Is Not Null**
- **In <name of procedure that retrieves dynamically a list of authorized values>**

  See the *Dynamic Analytic Privileges* section.

Note:
All filter operators, except `IsNull` and `Is Not Null`, accept empty strings (" ") as filter operands. For example:

- *In (" ", "A", "B")*
- *Between (" ", "XYZ")* (as lower limit in comparison operators)

Activating Analytic Privileges

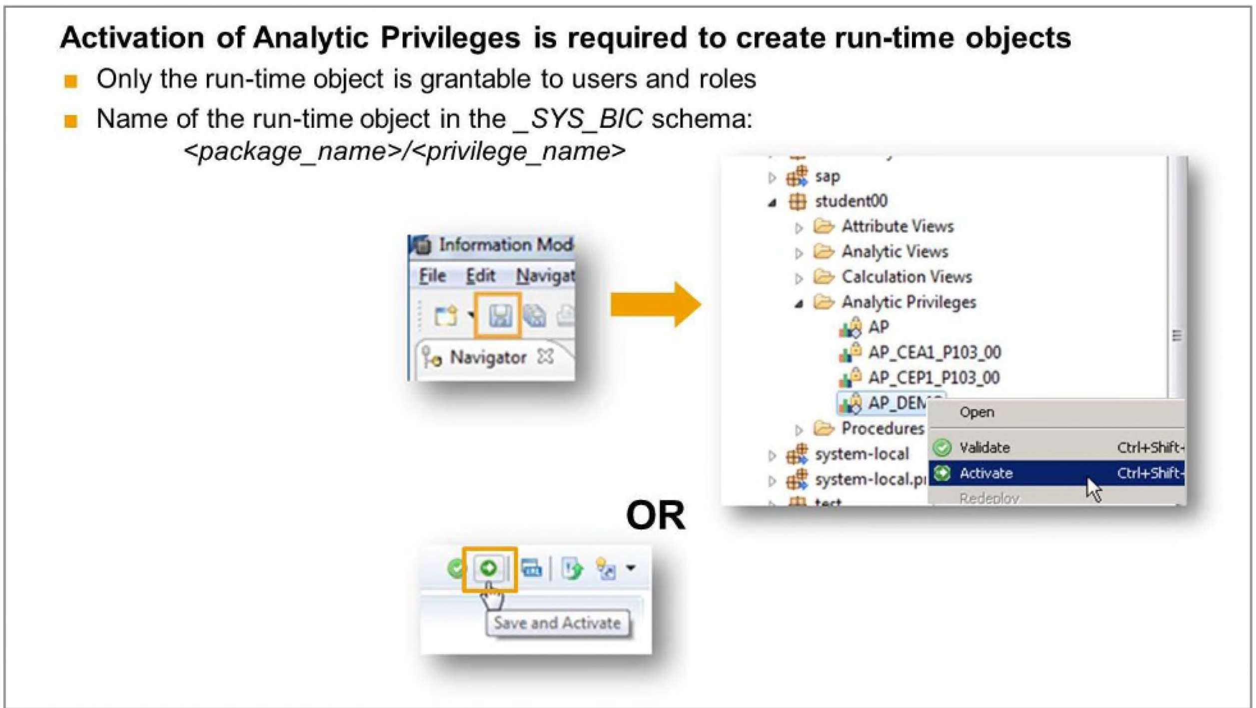


Figure 386: Activating Analytic Privileges

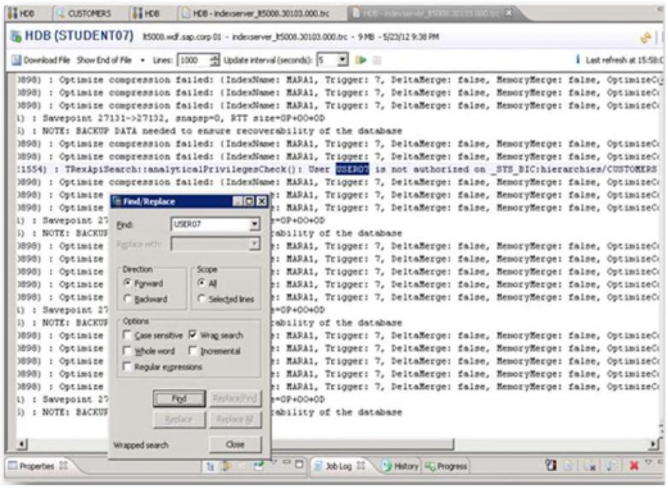
Like other information views, Analytic Privileges must be activated before they can be granted to the end-users.

The activation creates the run-time version of the Analytic Privilege in the *_SYS_BIC* schema.

Analytic Privilege Check

The Analytic Privilege Check evaluates Analytical Privileges:

- Granted to the User
- Applicable to the queried Information Model
- Currently valid (validity dates)
- With Attribute restrictions covering attributes of the view



- If no Analytic Privilege for the user can be found, user queries are rejected with a "…not authorized" error message.
- Detailed info can be found in the trace file of the Index Server (Administration → Diagnosis Files)

Figure 387: Analytic Privilege Check

The figure summarizes the Analytic Privilege check process.

Caution:
SAP HANA includes a built-in Analytic Privilege, called *_SYS_BI_CP_ALL*, which allows access to all the data of any Information View (except those designed to check SQL Analytic Privileges). That is, it overrules any other Classical Analytic Privilege and gives access to the full data set of any view designed to check Classical Analytic Privileges.

- This privilege, which is extremely extended, should not be granted to standard end-users, especially in a productive system.
- Moreover, this privilege is, by default, granted to the standard role *MODELING*. For this reason, this role must not be assigned directly to end-users, but it can be used as a template.

If you need to solve issues related to the behavior of Analytic Privileges, you can find detailed information in the trace file of the Index Server:

- In the context menu of the SAP HANA System, choose *Administration* and select the *Diagnosis Files* tab.
- Locate the file *index server_<host name>.<port>.<counter>.trc* and double-click to open it.
- Choose *Show End of File* and search for the user name (press Ctrl+ F).

The trace log displays the Analytical Privilege Check errors.

SQL Analytic Privileges

SQL Analytic Privileges have been introduced with SAP HANA SPS10.

They provide the following benefits:

Key Benefits of SQL Analytic Privileges.

- High flexibility to support complex filtering scenarios, in a familiar SQL environment
- Possibility to define attribute restrictions based on hierarchies, by selecting a hierarchy node
- Possibility to create Analytic Privileges in two steps:
  - First, by choosing Attributes and defining Restrictions for each of them, like in a Classical Analytic Privilege.
  - Then (optionally), by converting the corresponding restrictions into an SQL Statement, and adding SQL-based custom restrictions.
- Possibility to secure data access to Core Data Services (CDS) artifacts, in particular CDS Views.

> Note:
> The type of Analytic Privilege must generally be chosen when you create it.
>
> However, to convert Classic Analytic Privileges to SQL Analytic Privileges, the *Migrate* utility can be used.

Securing CDS Artifacts with SQL Analytic Privileges

From SAP HANA SPS11 onwards, it is possible to apply SQL Analytic Privileges to CDS Artifacts.

This offers a new possibility in implementation scenarios where CDS Views are used and exposed to the end-user.

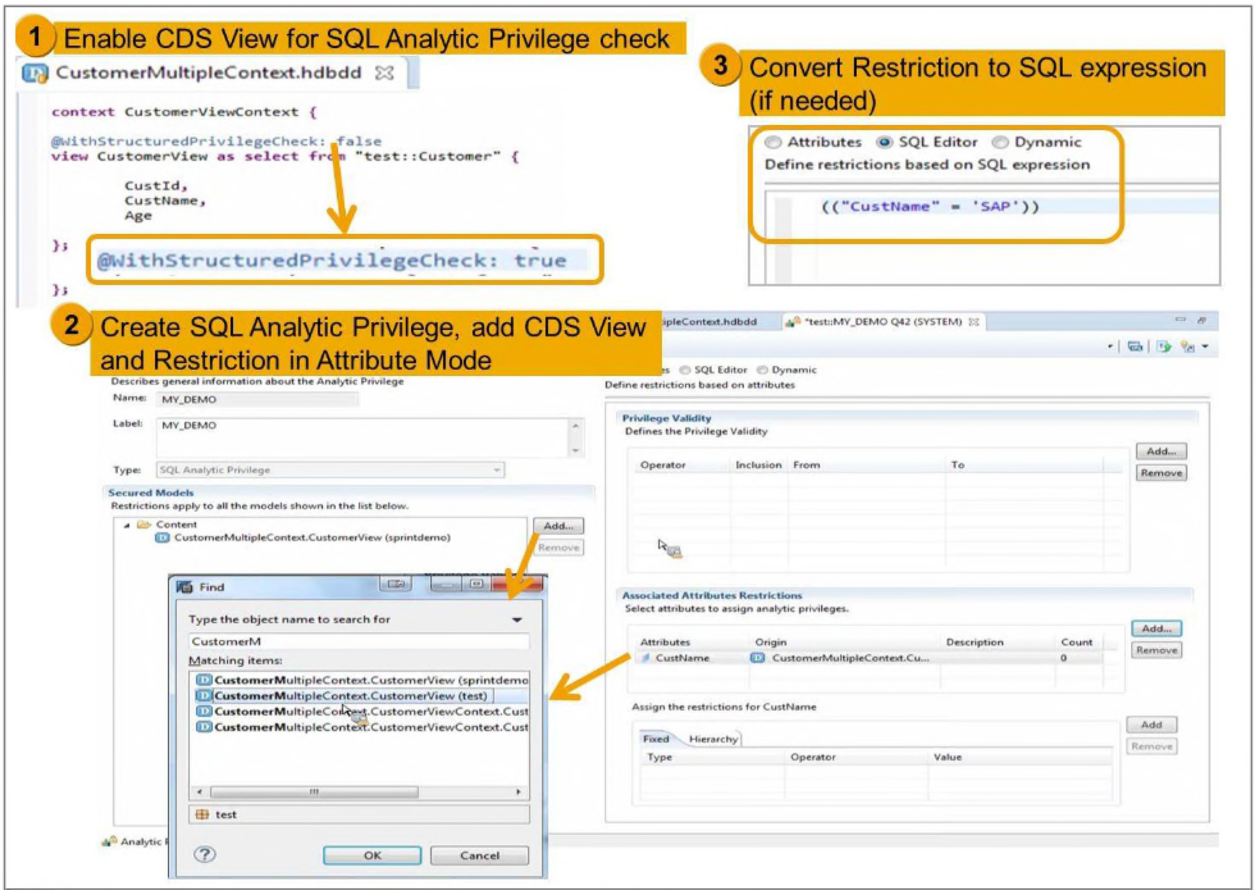


Figure 388: Securing CDS Views with SQL Analytic Privileges Image

Defining Restrictions on Hierarchies Nodes in SQL Analytic Privileges

SAP HANA SPS11 supports attribute restrictions based on a hierarchy node (rather than a list of values, and interval) in SQL Analytic Privileges.

This is useful in scenarios where a user who is enabled to a certain node of the hierarchy must also be enabled to all the descendants of this node.

With a geographical hierarchy, for example, the Manager of the North America area will see all the countries in his area if the Analytic Privilege sets the attribute restriction to the *NA* node. You do not have to list all the countries from the area.

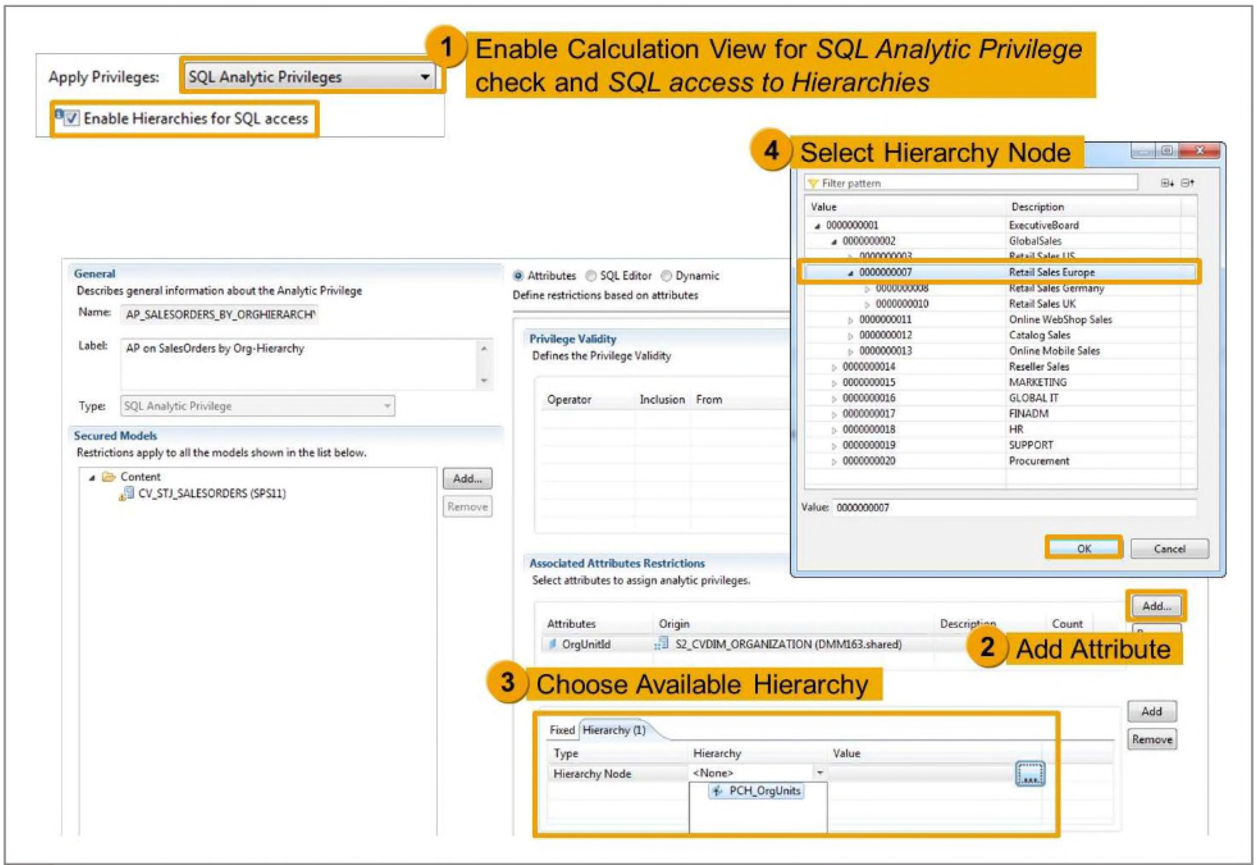


Figure 389: Hierarchy-Based Restrictions in SQL Analytic Privileges

> Caution:
> These features are supported:
>
> - In SQL Analytic Privileges (NOT Classic ones).
> - With Calculation Views of type *CUBE With Star Join* that are designed to check SQL Analytic Privilege and enable SQL access to hierarchy.
> - For Parent-Child Hierarchies.

Dynamic Analytic Privileges

In an Analytic Privilege, in addition to static values filtering conditions, it is possible to determine the filtering conditions using a Catalog or Repository Procedure. The Analytic Privilege is called in this case a *Dynamic Analytic Privilege*.

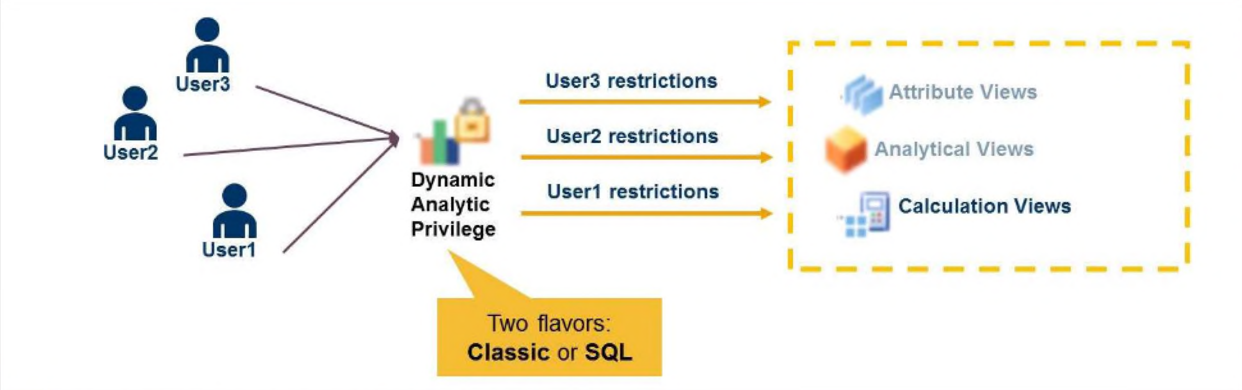


Figure 390: Dynamic Analytic Privileges - Concept

With this approach, the filtering conditions that apply for a specific user are determined at runtime, when querying a specific view. This allows a more scalable approach, where the same Analytic Privilege can be granted to several users who having different authorization requirements.

For example, the stored procedure used in the Analytic Privilege can query the user's business department from a table representing the company's structure, and check in another table whether the user has a manager role.

Classical vs. SQL Dynamic Analytic Privileges

Both types of Analytic Privileges types, Classic and SQL, can be designed as Dynamic Analytic Privileges, that is, Analytic Privileges that use Procedures. However, the way to implement them is very different.

Using Procedures in Dynamic Analytic Privileges

- In a Classic Analytic Privilege
  - Each attribute can have a dynamic definition based on a procedure.
  - You can combine fixed and procedure-based filtering conditions on different attributes (for example, a fixed range of dates and a procedure-based range of countries or products).
- In a SQL Analytic Privilege
  - Only one procedure is used by the Analytic Privilege.
  - This procedure must return a character string that is an SQL `WHERE` clause defining the different columns and the restricted values.

    For example, `"COUNTRY"='US' AND "YEAR"='2016'`.

Classical Dynamic Analytic Privilege - Designing the Underlying Procedures

The parameters of a procedure used in a Classical Dynamic Analytic Privilege must be defined as follows:

Classical Dynamic Analytic Privilege - Procedure Parameters

- No input parameters
- 1 output parameter of type *Table* with one single column, for the `In` operator
- 1 output parameter of a scalar type for all unary operators, such as `Equal`

- 2 output parameters of a scalar type for the binary operator `Between`

You can get detailed information on further restrictions that apply in the *SAP HANA Developer Guide* available on the SAP Help Portal.

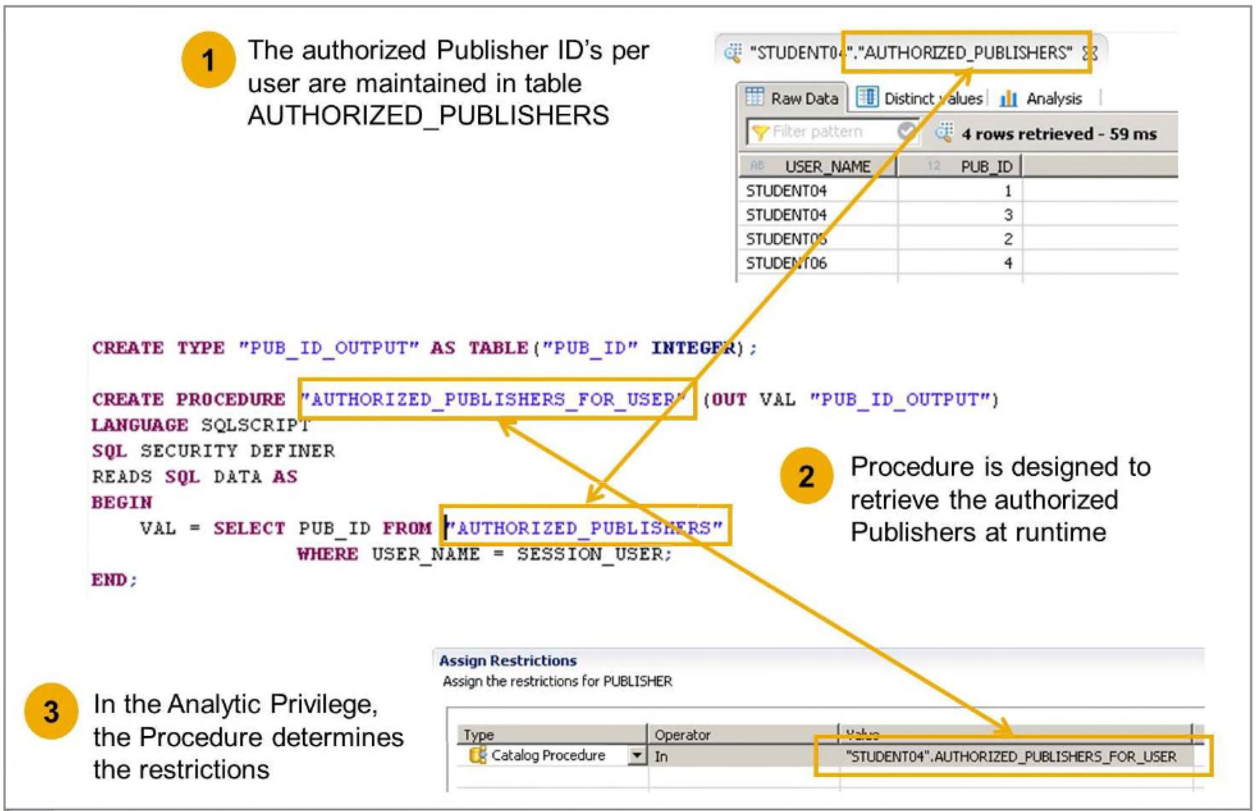


Figure 391: Classical Dynamic Analytic Privilege - Example

In the figure, Classical Dynamic Analytic Privilege - Example, a Dynamic Analytic Privilege is used to retrieve the list of publishers that each user is able to see.

> Note:
> The `In` operator can only be used in procedure-based restriction filters.
>
> For fixed restriction filters, however, expressing an `In` condition can be done with several *Equals* conditions.

Dynamic SQL Analytic Privilege - Designing the Underlying Procedures

A SQL Analytic Privilege only uses one underlying procedure, which is a valid SQLScript expression to filter data, like in the `WHERE` clause of a SQL statement.

This procedure must have the following properties:

SQL Analytic Privilege - Procedure Properties

- Procedure must be read-only
- Security mode must be *DEFINER*
- No input parameters
- Only one scalar output parameter of type VARCHAR(256) or NVARCHAR(256)

Defining Data Access Security with Nested Information Views

There are many business cases where Information Views contain references to one another.

The figures, Nested Information Views - Data Access Security Principles, explain how data access is handled when Information Views are nested.

For each view, the following criteria are considered:

- Does the user have `SELECT` privilege on the (activated) view, that is, the column view in the *_SYS_BIC* schema?
- Is the view designed to check Analytic Privileges?
- Is the user granted Analytic Privileges for the view?

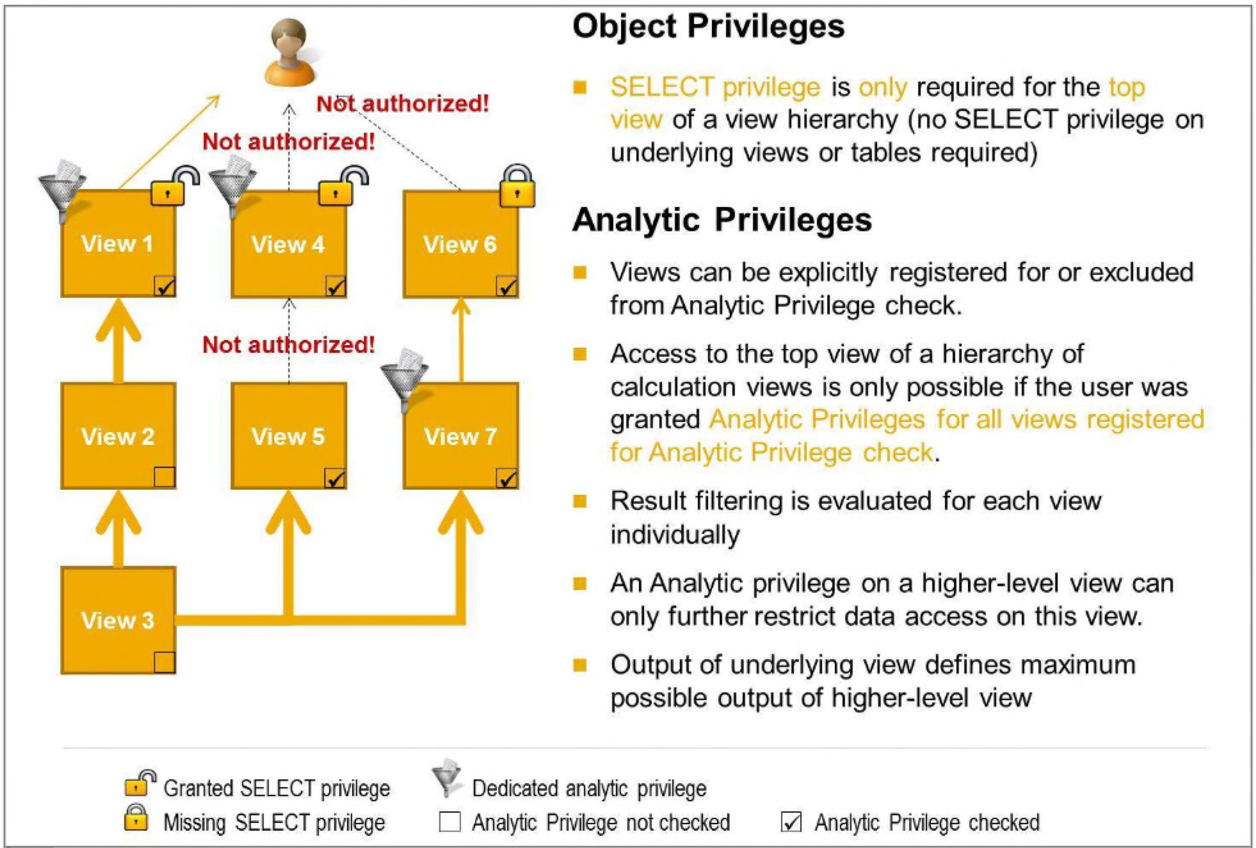


Figure 392: Nested Information Views - Data Access Security Principles

The key rules that govern the access to data are as follows:

- Object Privileges

  There is no need to grant `SELECT` privileges on the underlying views or tables. The end user only needs to be granted `SELECT` privileges on the top view of the view hierarchy.

- Analytic Privileges

  The Analytic Privileges logic is applied through all the view hierarchy.

  Whenever the view hierarchy contains at least one view that is checked for Analytic Privileges but for which the end user has no Analytic Privilege, no data is retrieved (not authorized).

> Note:
> In the figure, Nested Information Views - Data Access Security Principles, this is the case for View 5, so View 4 will not retrieve any data.
>
> This is also the case for View 6; however, it was not available because SELECT privilege is missing.

Data Access Security in SAP HANA Live

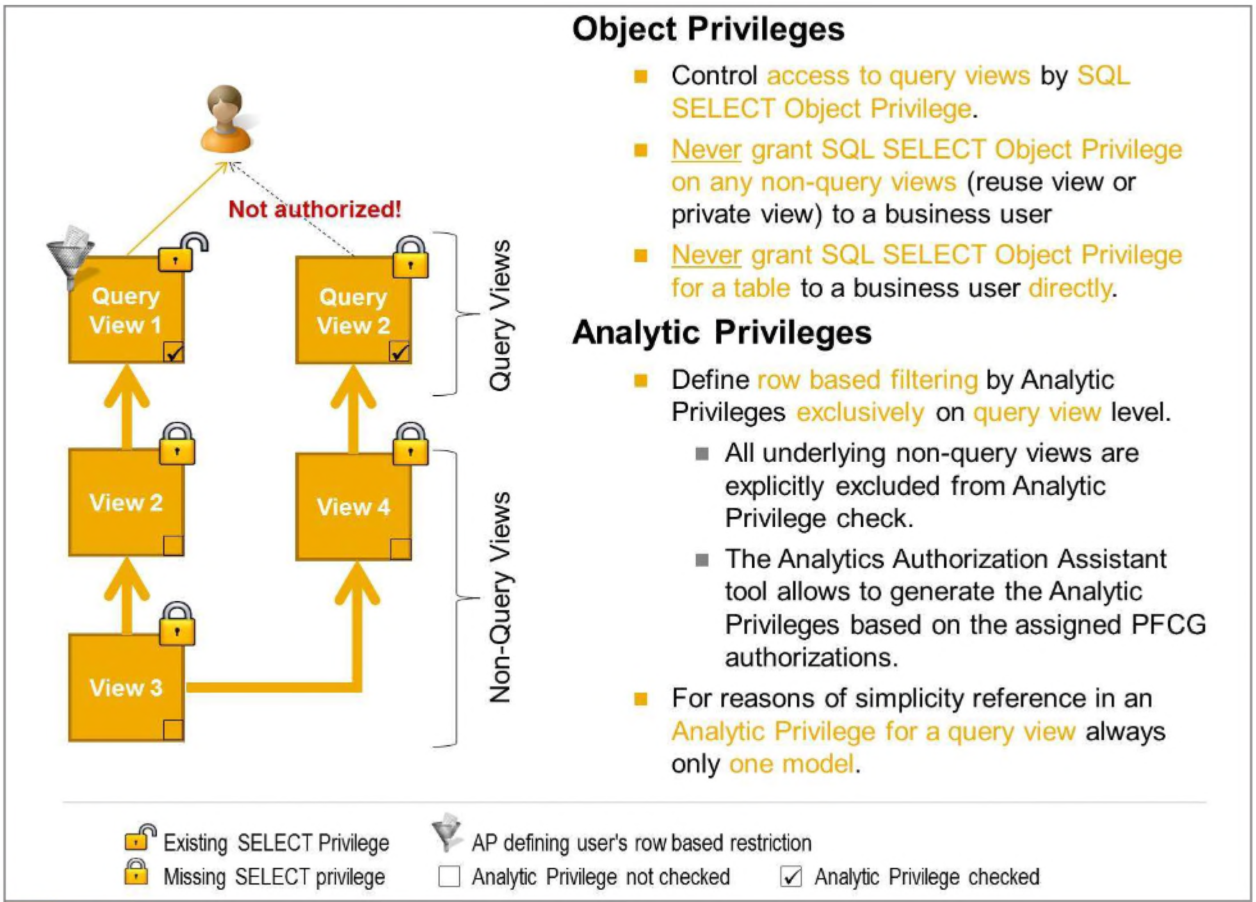


Figure 393: Data Access Security in SAP HANA Live

The SAP HANA Live Virtual Data Model relies heavily on views nesting, and in addition, classifies views as *Query Views* (exposed to the end-user) and non-Query Views (namely, *Private* and *Reuse Views*) which should not be accessed directly by the end user.

The Data Access Security of SAP HANA Live is implemented as follows:

- The end users are granted the Object Privilege SELECT exclusively on *Query Views*. They are never granted any privilege on non-Query views or database tables.
- By design, *Query Views* perform a check for Analytic Privileges and Analytic Privileges must be defined on these views to enable end-users to see the data.

  The non-Query views do not perform any check for Analytic Privileges, which is only possible, from a security perspective, because end-users cannot access them (missing SELECT Privileges on the views).

Managing Authorizations on Activated Content

In SAP HANA, the activated version of any repository objects is owned by the technical user *_SYS_REPO*.

For this reason, specific procedures exist in SAP HANA to grant or revoke privileges related to activated content. This is like granting or revoking privileges on behalf of the user *_SYS_REPO*.

These procedures can be called by SQLScript from the SQL Editor, but they are also called transparently when you open a user (resp. a role), and grant or revoke authorizations on these types of objects to the user (resp. the role).

> Note:
> The procedures are specific to Repository Objects owned by *_SYS_REPO*.
>
> You cannot use them to manage authorizations on classic catalog objects, such as tables created by a *CREATE TABLE* SQL Statement.

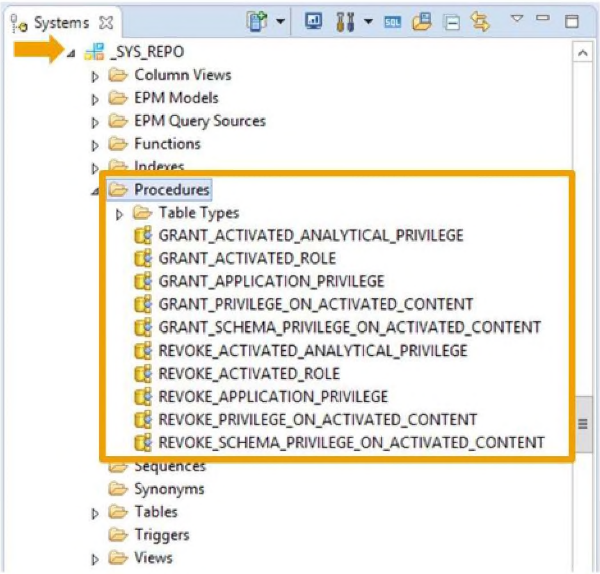


Figure 394: Managing Authorizations on Activated Content

Using Procedures to Grant or Revoke Privileges

The procedures allow you to grant or revoke the following types of privileges:

Table 23: Procedures for Managing Authorizations on Repository Objects

Type of Repository Object	Procedure name
Analytic Privileges	GRANT_ACTIVATED_ANALYTICAL_PRIVILEGE REVOKE_ACTIVATED_ANALYTICAL_PRIVILEGE
Repository Roles	GRANT_ACTIVATED_ROLE REVOKE_ACTIVATED_ROLE
Application Privileges	GRANT_APPLICATION_PRIVILEGE REVOKE_APPLICATION_PRIVILEGE

Type of Repository Object	Procedure name
Object Privileges (for example, on an information view, a procedure, a table)	GRANT_PRIVILEGE_ON_ACTIVATED_CONTENT REVOKE_PRIVILEGE_ON_ACTIVATED_CONTENT
Object Privileges (on a schema)	GRANT_SCHEMA_PRIVILEGE_ON_ACTIVATED_CONTENT REVOKE_SCHEMA_PRIVILEGE_ON_ACTIVATED_CONTENT

The procedures are called in a classical way, by passing the relevant parameters. For example, for an Object Privilege such as an activated table, the parameters are:

- Object privilege: `SELECT`, `EXECUTE`, and so on.
- Object name: the fully qualified name of the object.
- User/Role

  : the grantee.

To be able to call these procedures, you must be granted the `EXECUTE` privilege on them (or only some of them).

> Note:
> In the list of granted privileges for a specific user or role, the grantor of these privileges will always be *_SYS_REPO*.

Summary of Privileges Required for the Modeler and End-User

The figures, Summary of Required Privileges for the Modeler Role (1) and (2), highlight, for each action that is relevant for the Modeler (including data preview in SAP HANA Studio or Data Query with a reporting tool), the required privileges.

Front-End Tool	SAP HANA STUDIO						Reporting Tool
Object Type	Information View						Information View
Action	Open (Design)	Create/ Modify	Delete	Activate	Take Over	Data Preview	Query Data
SYSTEM PRIVILEGES							
REPO_WORK_IN_FOREIGN_WORKSPACE					x		
REPO_EXPORT							
REPO_IMPORT							
REPO_MAINTAIN_DELIVERY_UNITS							
OBJECTS PRIVILEGES							
EXECUTE on procedure SYS.REPOSITORY_REST	x	x	x	x	x	x	
SELECT on underlying tables	x (1)	x		o		o	o
SELECT on schema _SYS_BI				o		x	x
SELECT on activated queried view (8)						x	x
SELECT on activated underlying views (8)				o		o	o
PACKAGE PRIVILEGES							
REPO.READ	x	x (6)	x	x	x	x	o
REPO.EDIT_NATIVE_OBJECTS		x	x				
REPO.ACTIVATE_NATIVE_OBJECTS				x			
REPO.MAINTAIN_NATIVE_PACKAGES							
REPO.EDIT_IMPORTED_OBJECTS		(2)					
REPO.ACTIVATE_IMPORTED_OBJECTS				x (3)			
REPO.MAINTAIN_IMPORTED_PACKAGES			(2)				
ANALYTIC PRIVILEGES							
Valid AP Applicable to the Queried View						x (4)	x (4)
Valid AP Applicable to the Underlying Views						x (4)	x (4)
SYS_BI_CP_ALL						(5)	(5)

x **The Privilege is required**
o **The Privilege is NOT required (pay particular attention)**
[blank] **The Privilege is not required**

Figure 395: Summary of Required Privileges for the Modeler Role (1)

Summary of Required Privileges for the Modeler Role (2)

Front-End Tool	SAP HANA STUDIO					
Object Type	Package			Delivery Unit		
Action	List Models and sub-packages	Create sub-package	Delete sub-package	Export	Import	Modify
SYSTEM PRIVILEGES						
REPO_WORK_IN_FOREIGN_WORKSPACE						
REPO_EXPORT				x		
REPO_IMPORT					x	
REPO_MAINTAIN_DELIVERY_UNITS						x
OBJECTS PRIVILEGES						
EXECUTE on procedure SYS.REPOSITORY_REST	x	x	x	x	x	x
SELECT on underlying tables						
SELECT on schema _SYS_BI						
SELECT on activated queried view (8)						
SELECT on activated underlying views (8)						
PACKAGE PRIVILEGES						
REPO.READ	x	x	x			
REPO.EDIT_NATIVE_OBJECTS		x				
REPO.ACTIVATE_NATIVE_OBJECTS						
REPO.MAINTAIN_NATIVE_PACKAGES		x	x (7)			
REPO.EDIT_IMPORTED_OBJECTS						
REPO.ACTIVATE_IMPORTED_OBJECTS						
REPO.MAINTAIN_IMPORTED_PACKAGES						
ANALYTIC PRIVILEGES						
Valid AP Applicable to the Queried View						
Valid AP Applicable to the Underlying Views						
SYS_BI_CP_ALL						

x **The Privilege is required**
o **The Privilege is NOT required (pay particular attention)**
[blank] **The Privilege is not required**

Figure 396: Summary of Required Privileges for the Modeler Role (2)

Here is the additional information referenced in the tables:

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

1. Opening the view is possible, but the tables where a user has no privileges, appear as missing or not accessible.
2. You should not modify or delete imported objects, except in rare circumstances.
3. In some cases, it is necessary to validate the models of imported packages after importing the delivery unit.
4. Analytic Privilege needed only if the view is designed to test Analytic Privileges.
5. This is an extensive privilege that overrules any other Classical Analytic Privilege. Do NOT assign it to end-users.
6. *REPO.READ* also needed on any package containing views that you want to reference.
7. Privilege is required on the sub-package you delete, not on the containing package.
8. Column Views in schema *_SYS_BIC*

If you consider a user who wants to preview the data of an information view in SAP HANA studio or query this view with a reporting tool, the authorizations work as follows:

- In general, the user has access to an individual, independent view (Attribute, Analytic, or Calculation View) if the following prerequisites are met:
  - The user was granted the `SELECT` privilege on the view or the containing schema.
  - The user was granted an Analytic Privilege that is applicable to the view (if the view is designed to check Analytic Privileges).
- No `SELECT` privilege on the underlying base tables or views of this view is required.

When relevant Analytic Privileges are found for the current user and the selected view, the evaluation process ensures that, according to the value filters specified in the Dimension restrictions, the appropriate data is presented to the user.

Unit 7
Exercise 31

Create and Assign Analytic Privileges

Exercise Objectives

After you complete this exercise, you will be able to:

- Create a SQL Analytic Privileges
- Assign an Analytic Privilege to a user

Business Example

You are an SAP HANA developer and need to apply restrictions on the data that a user can view.

You will grant the user a generic permission to see all the data. Afterwards, you will restrict the data that the user is allowed to see by defining an Analytic Privilege.

> Caution:
> During this exercise, you will work with two different users:
>
> - *STUDENT##* creates the Calculation View, defines and assigns Analytic Privileges.
> - *USER##* is the user you're defining data access for. This user will only try to preview the view's data in SAP HANA Studio.
>
> Make sure you always perform the steps with the specified user. The user performing the action is determined by the entry in the *Systems* view you're navigating from.
>
> Each entry mentions the logged on user, you will typically navigate from *H00 (STUDENT##) SAP HANA* or *H00 (USER##) SAP HANA*.

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.

Task 1: Copy a Calculation View

During this exercise, as *STUDENT##*, you will work on your own view, which you first have to copy from the training package.

1. Copy the Calculation View CVCS_PO2 from the package *training* → *ha300* to your STUDENT## package.

   Name the copy CVCS_PO2_##.

2. Activate the Calculation View *CVCS_PO2_##*.

Task 2: Assign a Generic Analytic Privilege and Preview the Data

You will assign an generic Analytic Privilege that allows to view all the data to *USER##*.

> Caution:
> In the context of this exercise, you will assign the Analytic privilege to a specific user. Keep in mind that, in real life, you should most often assign privileges to roles (rather that individual users).

1. As *STUDENT##*, Assign the Analytic Privilege *_SYS_BI_CP_ALL* to *USER##*.

   > Caution:
   > With this Analytic Privilege, a user passes all Classical Analytic Privilege authorizations globally.
   >
   > In a productive environment, make sure you do not grant this privilege to any user.

2. If needed, connect to the same SAP HANA system *H00* with a different user: *USER##*.

   The password for this user is **`Training1`**.

   > Hint:
   > Right-click the *H00 (STUDENT##)* entry in the *Systems* view.

3. As *USER##*, preview the data of the Calculation View *CVCS_PO2_##*.

   Do you expect to be able to see all the data in the view?

Task 3: Change the Analytic Privileges Check Setting for the Calculation View

You will now change the setting of your Calculation View to check SQL Analytic Privileges, and then try to display again the view's data.

1. As *STUDENT##*, open the Calculation view *CVCS_PO2_##* and change its *Apply Privileges* property to *SQL Analytic Privileges*.

2. Save and Activate the Calculation View.

3. As *USER##*, preview the data of the Calculation View *CVCS_PO2_##*.

   What do you observe? What is your conclusion about the *SYS_BI_CP_ALL* Analytic Privilege?

Task 4: Create and Assign a New Analytic Privilege

You now want to restrict the authorized data for the view *CVCS_PO2_##* to the Business Partners from the US, for *USER##*.

After removing the previously assigned Analytic Privilege, you will create a new Analytic Privilege and assign it to *USER##*.

You will then preview the data in the calculation view again.

1. As *STUDENT##*, create a new SQL Analytic Privilege **`APS_COUNTRY_US_##`** (description: **`Country US`**) in order to restrict access on the Calculation View *CVCS_PO2_##*.

Set the restriction on *CVCS_PO2_##* for Attribute *COUNTRY* equals to *US*.

> Caution:
> Pay attention to work with the correct users:
>
> - *STUDENT##* is the role of the SAP HANA developer who is building the authorization concept.
> - *USER##* is the user who will be granted the authorizations.

2. Activate the new Analytic Privilege.
3. As *STUDENT##*, delete any existing Analytic Privilege for user *USER##*, and assign the newly created Analytic Privilege *APS_COUNTRY_US_##*.
4. As *USER##*, check the impact of assigning the newly created Analytic Privilege.

   Perform a Data Preview of Calculation View *CVCS_PO2_##*.

   For which countries can you see the Purchase Order data?
5. (Optional) Open the definition of the view *CVCS_PO2_##* and list the other Information Views it references.

   What is your conclusion about the views *CV_PD and CV_BP2* with regards to their *Apply Privileges* setting?

Unit 7
Solution 31

Create and Assign Analytic Privileges

Exercise Objectives

After you complete this exercise, you will be able to:

- Create a SQL Analytic Privileges
- Assign an Analytic Privilege to a user

Business Example

You are an SAP HANA developer and need to apply restrictions on the data that a user can view.

You will grant the user a generic permission to see all the data. Afterwards, you will restrict the data that the user is allowed to see by defining an Analytic Privilege.

> Caution:
> During this exercise, you will work with two different users:
>
> - *STUDENT##* creates the Calculation View, defines and assigns Analytic Privileges.
> - *USER##* is the user you're defining data access for. This user will only try to preview the view's data in SAP HANA Studio.
>
> Make sure you always perform the steps with the specified user. The user performing the action is determined by the entry in the *Systems* view you're navigating from.
>
> Each entry mentions the logged on user, you will typically navigate from *H00 (STUDENT##) SAP HANA* or *H00 (USER##) SAP HANA*.

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.

Task 1: Copy a Calculation View

During this exercise, as *STUDENT##*, you will work on your own view, which you first have to copy from the training package.

1. Copy the Calculation View CVCS_PO2 from the package *training* → *ha300* to your STUDENT## package.

   Name the copy CVCS_PO2_##.

a) In the *Systems* view, expand the *Content* folder to display the package *training → ha300 → Calculation Views*.

b) Right-click the Calculation View CVCS_PO2 and choose *Copy*.

c) Right-click your package *STUDENT##* and choose *Paste*.

d) In the *Copy Object* dialog box, enter **CVCS_PO2_##** and choose *OK*.

2. Activate the Calculation View *CVCS_PO2_##*.

a) Right-click the view *CVCS_PO2_##* in your *STUDENT##* package and choose *Activate*.

b) Check the outcome of the Activation job in the *Job Log* view.

Task 2: Assign a Generic Analytic Privilege and Preview the Data

You will assign an generic Analytic Privilege that allows to view all the data to *USER##*.

> Caution:
> In the context of this exercise, you will assign the Analytic privilege to a specific user. Keep in mind that, in real life, you should most often assign privileges to roles (rather that individual users).

1. As *STUDENT##*, Assign the Analytic Privilege *_SYS_BI_CP_ALL* to *USER##*.

> Caution:
> With this Analytic Privilege, a user passes all Classical Analytic Privilege authorizations globally.
>
> In a productive environment, make sure you do not grant this privilege to any user.

a) In the *Systems* view, under the system entry for *STUDENT##*, navigate to the *Security → Users* folder.

b) Double-click the user *USER##*.

c) Select the *Analytic Privileges* tab.

d) To add an Analytic Privilege, choose *(Add Analytic Privilege)*.

e) In the *Select Analytic Privilege* screen, type the first characters of the string **_SYS_BI_CP_ALL** and when the Analytic Privilege appears in the list, select it and choose *OK*.

f) To deploy the user *USER##*, press F8 or choose *(Deploy)*.

2. If needed, connect to the same SAP HANA system *H00* with a different user: *USER##*.

The password for this user is **Training1**.

> Hint:
> Right-click the *H00 (STUDENT##)* entry in the *Systems* view.

a) In the *Systems* view, right-click the existing system connection and choose *Add System with Different User*.

b) Enter the user name and password.

c) Choose *Finish*.

3. As *USER##*, preview the data of the Calculation View *CVCS_PO2_##*.

Do you expect to be able to see all the data in the view?

a) In the *Systems* view, under the System for *USER##*, navigate to *Content → STUDENT## → Calculation Views*.

b) Right-click the Calculation View *CVCS_PO2_##* an choose *Data Preview*,

c) Select the *Raw Data* tab.
The result displays all data, because now your user has the Analytic Privilege *_SYS_BI_CP_ALL*.

Task 3: Change the Analytic Privileges Check Setting for the Calculation View

You will now change the setting of your Calculation View to check SQL Analytic Privileges, and then try to display again the view's data.

1. As *STUDENT##*, open the Calculation view *CVCS_PO2_##* and change its *Apply Privileges* property to *SQL Analytic Privileges*.

a) In your *STUDENT##* package, double click the view *CVCS_PO2_##*.

b) In the *Semantics* node, display the *View Properties* tab.

c) In the *Apply Privileges* drop-down list, select *SQL Analytic Privileges*.

2. Save and Activate the Calculation View.

a) Choose *Save and Activate*.

b) Check the outcome of the Activation job in the *Job Log* view.

3. As *USER##*, preview the data of the Calculation View *CVCS_PO2_##*.

What do you observe? What is your conclusion about the *SYS_BI_CP_ALL* Analytic Privilege?

a) In the *Systems* view, under the System for *USER##*, navigate to *Content → STUDENT## → Calculation Views*.

b) Right-click the Calculation View *CVCS_PO2_##* and choose *Data Preview*,

c) Select the *Raw Data* tab.
You cannot see any data and get an error *Insufficient privilege*. The generic privilege *SYS_BI_CP_ALL* by-passes only Classical Analytic Privileges. You cannot access data from Information Views secured by SQL Analytic Privileges with this generic privilege. Instead, you always need one or several relevant Analytic Privileges.

Task 4: Create and Assign a New Analytic Privilege

You now want to restrict the authorized data for the view *CVCS_PO2_##* to the Business Partners from the US, for *USER##*.

After removing the previously assigned Analytic Privilege, you will create a new Analytic Privilege and assign it to *USER##*.

You will then preview the data in the calculation view again.

1. As *STUDENT##*, create a new SQL Analytic Privilege **`APS_COUNTRY_US_##`** (description: **`Country US`**) in order to restrict access on the Calculation View *CVCS_PO2_##*.

   Set the restriction on *CVCS_PO2_##* for Attribute *COUNTRY* equals to *US*.

   > Caution:
   > Pay attention to work with the correct users:
   >
   > - *STUDENT##* is the role of the SAP HANA developer who is building the authorization concept.
   > - *USER##* is the user who will be granted the authorizations.

   a) In the *Systems* view, under the entry for *STUDENT##*, right-click your *STUDENT##* package and choose *New → Analytic Privilege*.

   b) Enter the following name: **`APS_COUNTRY_US_##`** and choose *Next*.

   c) Select the Calculation View *CVCS_PO2_##* under your package *STUDENT##* and choose *Add*.

   d) Choose *Finish*.

   e) Under *Associated Attributes Restriction*, choose *Add*.

   f) In the *Select Object* window, select *COUNTRY* and choose *OK*.

   g) Under *Assign Restrictions*, choose *Add*.

   h) In the new restriction row, select the type *Fixed* and the operator *Equal*.

   i) In the *Value* column, choose the ellipsis *(...)* button to open the *Value Help* dialog and select *US*.

   j) Choose *OK*.

2. Activate the new Analytic Privilege.

   a) Choose `Save and Activate`.

3. As *STUDENT##*, delete any existing Analytic Privilege for user *USER##*, and assign the newly created Analytic Privilege *APS_COUNTRY_US_##*.

   a) In the *Systems* view, under the *STUDENT##* system entry, navigate to the folder *Security → Users*.

   b) Double-click *USER##*.

   c) In the *Analytic Privileges* tab, select the Analytic Privilege *_SYS_BI_CP_ALL* and choose the red *(X)* button.

   d) Choose *(Add Analytic Privilege)*.

   e) In the search bar, find *APS_COUNTRY_US_##*, select it and choose *OK*.

   f) To deploy the user *USER##*, press F8 or choose *(Deploy)*.

4. As *USER##*, check the impact of assigning the newly created Analytic Privilege.

   Perform a Data Preview of Calculation View *CVCS_PO2_##*.

For which countries can you see the Purchase Order data?

a) In the *Systems* view, under the system entry for *USER##*, navigate to your *STUDENT##* package. *Content* → *STUDENT##* → *Calculation Views*.

b) Right-click Calculation View *CVCS_PO2_##* and choose *Data Preview*.

c) Select the *Raw Data* tab.
Now, you should only see the Purchase Order data relating to Business Partners located in the US.

d) To confirm this, you can select the *Distinct Values* tab, and in the *Available Objects* pane, select the Attribute *COUNTRY*.
Only one attribute member, *US*, should be displayed.

5. (Optional) Open the definition of the view *CVCS_PO2_##* and list the other Information Views it references.

What is your conclusion about the views *CV_PD and CV_BP2* with regards to their *Apply Privileges* setting?

a) In your package, double-click the view *CVCS_PO2_##*.

b) Note that the Star Join node references two other DIMENSION Calculation Views: *CV_PD* and *CV_BP2*.

c) *USER##* can view the data without being granted any Analytic Privilege on these DIMENSION Calculation Views. You can conclude that *CV_PD* and *CV_BP2* are not set to apply Privileges (the *Apply Privileges* setting is blank).

Otherwise, you could only see data from the view *CVCS_PO2_##* after being granted relevant Analytic Privileges applicable to these views.

LESSON SUMMARY

You should now be able to:

- Define privileges
- Explain how to grant the relevant privileges to the Modeler and End-User

Unit 7
Lesson 3

Generating SAP HANA Live Privileges

LESSON OVERVIEW

With SAP HANA Live it is possible to generate privileges from ABAP authorizations. In this lesson we will explain how this is done.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Generate SAP HANA Live Privileges

Generating SAP HANA Live Privileges from SAP BusinessSuite

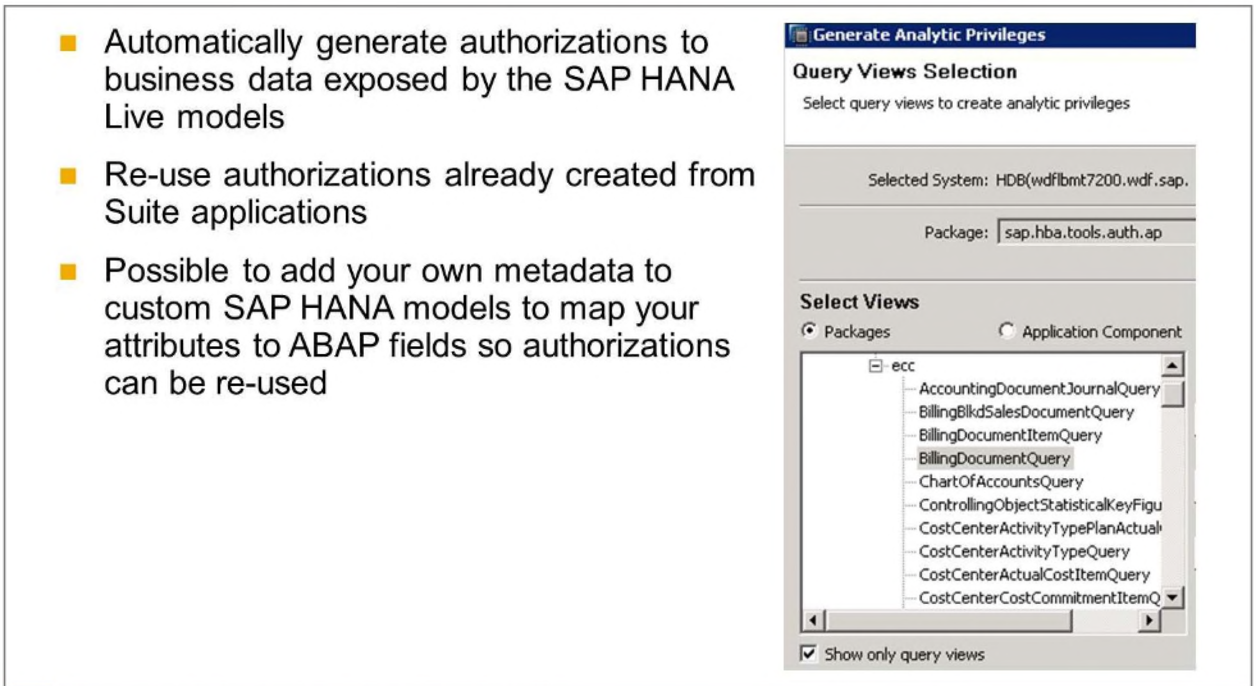


Figure 397: Authorization Assistant — Outline

With the SAP HANA Live Authorization Assistant, you can provide authorizations to users in the SAP HANA system that are required to access business data displayed by the virtual data model of SAP HANA Live.

SAP HANA Live Authorization Assistant takes those permissions into account that the same users already have in an ABAP-based Business Suite application.

Although this tool is a member of the SAP HANA Live family and is used to develop privileges for SAP HANA Live views, it can also be used to generate privileges for any SAP HANA views. The key requirement is that your information view contains analytic metadata (mappings between attribute names and ABAP authorization fields).

You manually add your own analytics metadata to any SAP HANA information view. Look in the properties of a view and there you will see a tab Analytics MetaData. Now just enter the ABAP authorization fields against your attribute names.

Authorization Assistant — Rationale

- Massively reduced manual effort in creating authorizations
- SAP supplies all metadata mapping between ABAP fields and SAP HANA attributes for all SAP HANA Live models
- Also works with customer created models and customer created ABAP authorizations (you supply the metadata mapping)
- Ensures user access consistency access across OLTP and OLAP environments

Figure 398: Authorization Assistant — Rationale

You can select multiple query views for multiple users and create analytic privileges for all the query views. You do not need to manually check for privileges in the SAP ABAP system and manually create privileges for each query view. Therefore, the mass process available with this tool reduces the effort required to create SAP HANA analytic privileges for information views. The existing analytic privileges can be reused between different users.

The Authorization Assistant can be used to update the SAP HANA analytic privileges whenever an ABAP authorization changed. This means you can easily keep the authorizations on the ABAP side in sync with the SAP HANA side. Then you can be sure users can only access the data from reports and applications that are based directly on SAP HANA that they are allowed to access from the suite applications.

The Authorization Assistant must be installed, and the detailed steps for installation are provided in the standard product documentation at **help.sap.com**.

An easy way to check if the tool is already installed is to find out if the SAP HANA studio menu option *Analytics Authorisation* is present. This menu option is added during the install process for the Authorization Assistant.

See SAP Note 1796718 for more details on this tool.

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

Using SAP HANA Live Analysis Authorization Assistant

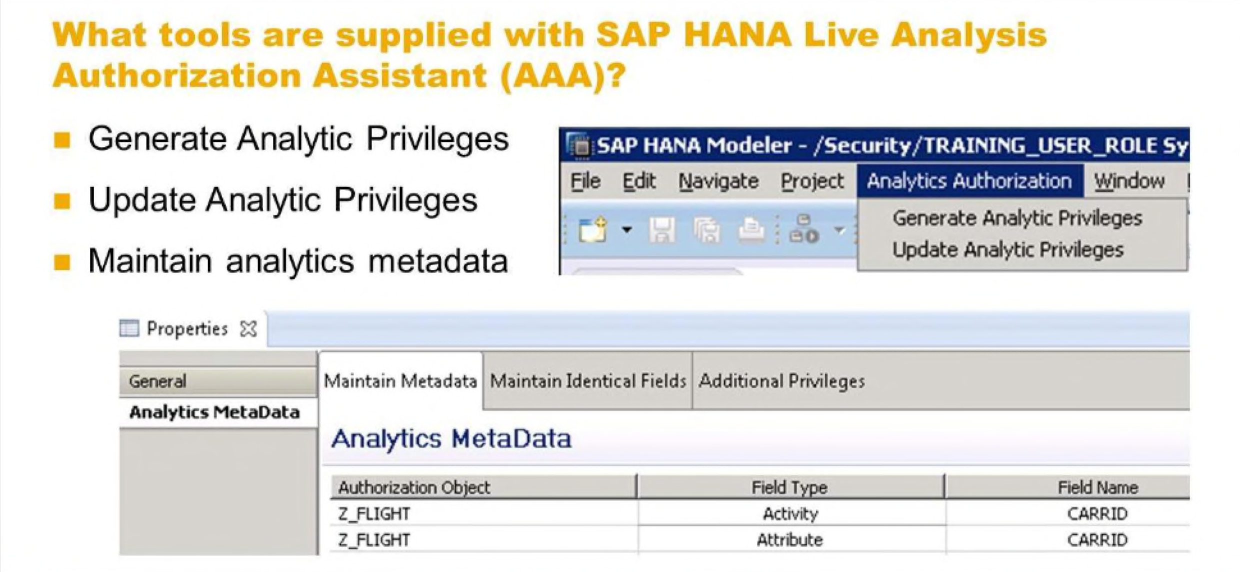


Figure 399: SAP HANA Live Analysis Authorization Assistant Tools

The main tools available with SAP HANA Authorization Assistant (often referred to as 'AAA') are downloaded from service marketplace as follows:

- Generate Analytic Privileges (this also includes Update Privileges function)
- Maintain Analytics Metadata

Generate Analytic Privileges

To generate SAP HANA analytic privileges follow the steps in the figure, Generate Analytic Privileges.

1. Open tool Generate Analytic Privileges
2. Select target SAP HANA system to receive generated privileges
3. Select the required SAP HANA Live views
4. Select ABAP user(s) from where Authorizations will be copied *
5. Review proposed privileges that will be generated
6. Accept and generate privileges
7. Assign generated privileges to SAP HANA user / roles **

Figure 400: Generate Analytic Privileges

* If you have selected views that use tables from multiple SAP HANA schemas, you can then select a schema in this step from where the user authorizations will be taken.

** A role is automatically generated with the name `ROLE_<ABAP user name>` and the generated privilege is automatically assigned to this role. If this role already exists (from a previous generation) the new privilege will be added to the role.

** Remember, the generated role is not automatically assigned to any SAP HANA user, this must be done manually.

Update Analytic Privileges

Do not manually modify any analytic privilege or roles generated by the tool.

However, what happens when changes are subsequently made to the ABAP authorizations? What does that mean to the SAP HANA privileges?

With the SAP HANA Live Authorization Assistant, you can re-synchronize analytic privileges that were previously generated with SAP HANA Live Analytics Authorization Assistant, with changes in the ABAP authorizations. The updated analytic privilege tool identifies the changes in the ABAP authorizations and new restrictions are created when you run the tool.

The basic steps are listed in the figure, Update Analytic Privileges.

1. Open tool Update Analytic Privileges
2. Select SAP HANA system in which you want to update privileges
3. Select the schema and ABAP users you want to check for updates
4. Select ABAP user(s) from where Authorizations will be copied *
5. Review proposed privilege updates
6. Accept and update privileges

Figure 401: Update Analytic Privileges

The valid analytic privileges are retained in the role and newly created analytic privileges are added. If the analytic privilege is not valid, it is removed from the role and if analytic privilege is not assigned to any role, it is deleted. The tool only checks if the analytic privilege is assigned to the role.

Adding Custom Authorization Metadata

SAP delivers the required metadata for all the relevant query views of the virtual data model. To view the SAP-delivered metadata, open the respective query view and navigate to *Properties → Analytics Metadata → Maintain Metadata*.

Features

- For customer developed views there is no mapping to ABAP source fields from the referenced tables
- It is possible to add your own field mappings to each attribute for your own SAP HANA views
- This meta-data can then be used to generate SAP HANA privileges from the ABAP source
- You cannot maintain analytics Authorization meta-data for SAP supplied views

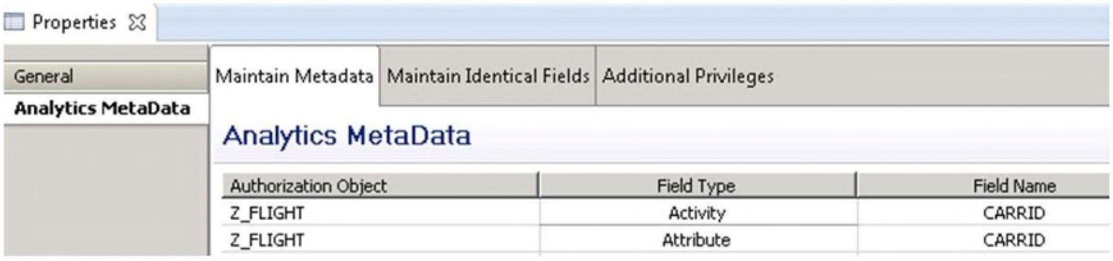


Figure 402: Adding Custom Authorization Metadata

In addition, you can use this tool to maintain metadata for custom views created that use tables from the ERP system.

You can add more rows by choosing + to map your own attributes to ABAP source fields.

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

Unit 7
Exercise 32

Generate an Analytic Privilege from an ABAP System

Business Example

You have been asked to generate an analytic privilege from an existing ABAP authorization.

> Note:
> This activity will be performed as a simulation that you can either watch as a demo, or work through the steps manually. The steps are fully described within the simulation so you should not need to refer to the exercise or solution provided. To get started follow these instructions:
>
> 1. In Windows Explorer, open your student folder *HA300 → HA300 Ex 32 - Generate an Analytic Privilege from an ABAP System*.
> 2. Double-click the file *index.html* to launch the simulation.
> 3. Choose either *DEMO* (just watch) or *PRACTICE* (you do the work), depending on how you want to proceed.
>
> The two letters ## represent the two-digit group number assigned to you by the instructor.

1. Deploy a new user **USER##**, set the initial password to **Training1** and assign the user the role *RESTRICTED_ROLE*.
2. Add another system to the *Systems View* assigning your new user.
3. Perform a data preview of the calculation view *SalesOrderQuery* What problem do you observe?
4. Review the ABAP authorizations that are currently assigned to user *authsales-##* that were generated from the role *HA900SALES* and make note of the values assigned to the object *V_VBAK_VKO*.
5. Generate an SAP HANA analytic privilege from the ABAP user AUTHSALES-## and ensure that the new analytic privilege has been assigned the same Sales Organization value as the ABAP authorization.
6. View the analytics metadata supplied with the standard SAP HANA live view *SalesOrderQuery*.
7. Assign the newly generated Analytic Privilege to the SAP HANA user *USER##* then perform another data preview to ensure the user is able to view only Sales Organization *1000* for the view *SalesOrderQuery*.

Generate an Analytic Privilege from an ABAP System

Business Example

You have been asked to generate an analytic privilege from an existing ABAP authorization.

> Note:
> This activity will be performed as a simulation that you can either watch as a demo, or work through the steps manually. The steps are fully described within the simulation so you should not need to refer to the exercise or solution provided. To get started follow these instructions:
>
> 1. In Windows Explorer, open your student folder *HA300 → HA300 Ex 32 - Generate an Analytic Privilege from an ABAP System*.
> 2. Double-click the file *index.html* to launch the simulation.
> 3. Choose either *DEMO* (just watch) or *PRACTICE* (you do the work), depending on how you want to proceed.
>
> The two letters ## represent the two-digit group number assigned to you by the instructor.

1. Deploy a new user **`USER##`**, set the initial password to **`Training1`** and assign the user the role *RESTRICTED_ROLE*.

   a) In the *Systems* view, expand the node *Security → Users* and choose *New User*.

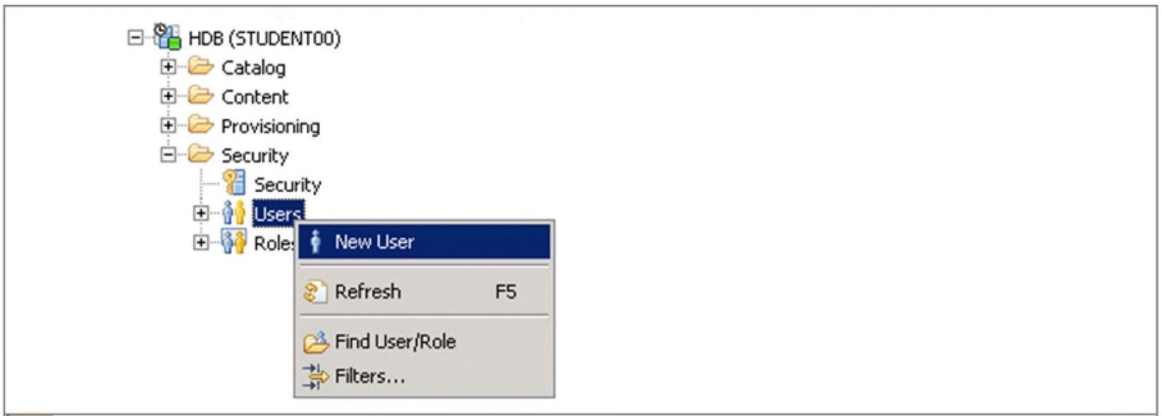


Figure 403: Users Node

   b) In the *User Name* field, enter **`USER##`**, and in the *Password* field, enter **`Training1`**. Do the same for the *Confirm* field.

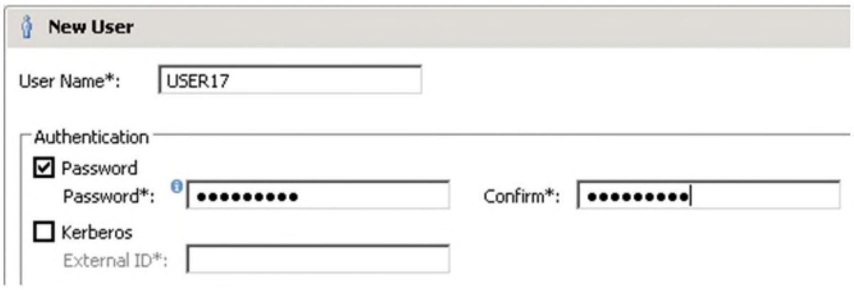


Figure 404: New User Dialog Box

c) Select the *Granted Roles* tab and choose +.



Figure 405: Grant Roles Tab

d) Locate *RESTRICTED_ROLE* using the search field. Highlight the role and choose *OK*.

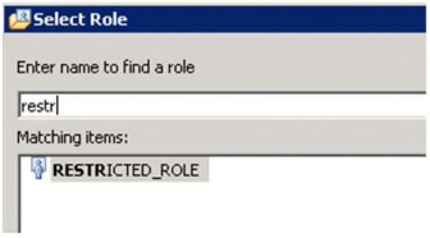


Figure 406: Select Role

e) Choose *Deploy*.

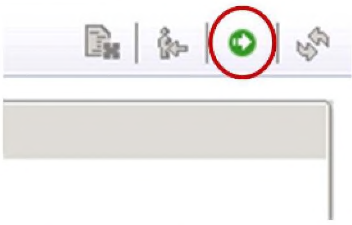


Figure 407: Deploy Button

2. Add another system to the *Systems View* assigning your new user.

a) Right-click on the instance node for *STUDENT##* and choose *Add Additional User*.

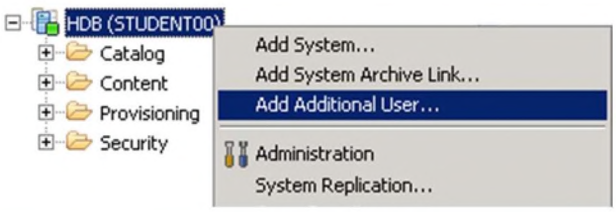


Figure 408: Student## Instance Node

b) Enter the user **`USER##`** and the password **`Training1`** then choose *Finish*.



Figure 409: Connection Properties

3. Perform a data preview of the calculation view *SalesOrderQuery* What problem do you observe?

   a) Under the *USER##* instance, expand the *Contents* folder to reach the *Calculation Views* folder.

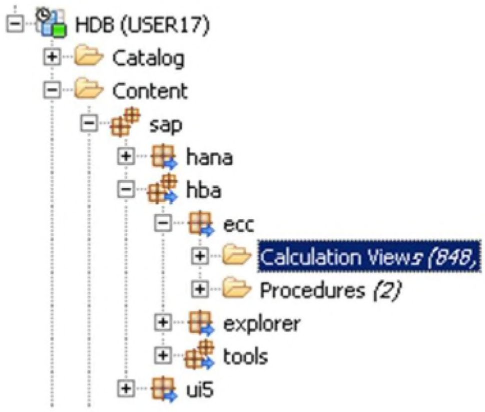


Figure 410: Calculation Views Folder

   b) Find the *SalesOrderQuery* Calculation View, right-click the view, and select *Data Preview*.

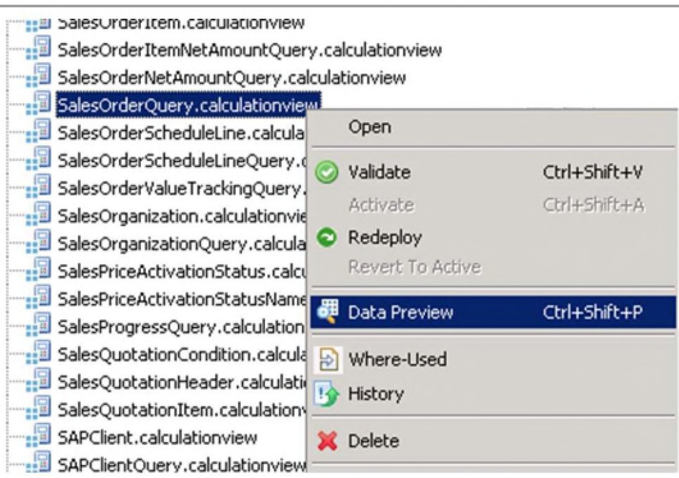


Figure 411: SalesOrderQuery Calculation View

c) At the prompt, in the *From* field, enter client **800**.

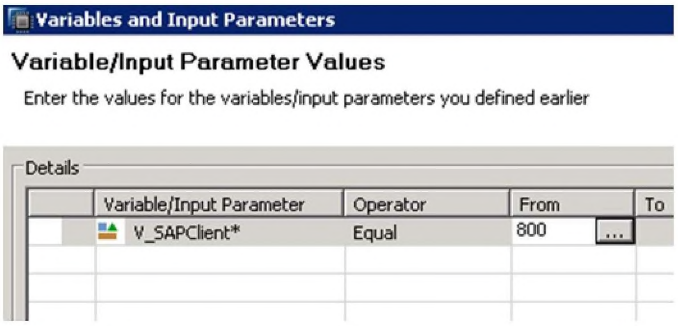


Figure 412: Variables and Input Parameters

d) An authorization error displayed because your user (USER##) does not have privileges to view any data in the view.

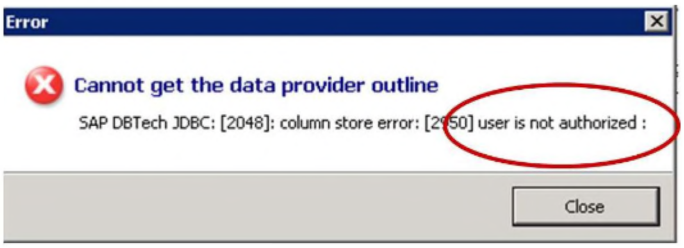


Figure 413: Error Message

4. Review the ABAP authorizations that are currently assigned to user *authsales-##* that were generated from the role *HA900SALES* and make note of the values assigned to the object *V_VBAK_VKO*.

   a) Log on to the ABAP system *ZME* using the following credentials:

      User: **student-##** (where ## is your number).

      Pass: **INITIAL** (you must change this the first time, choose any 6 characters).

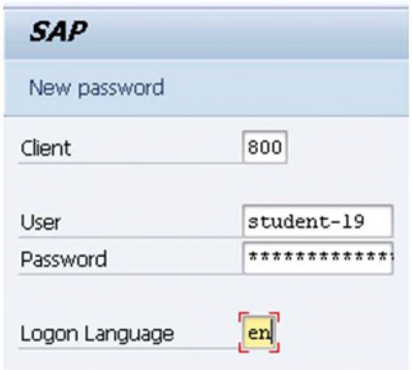


Figure 414: SAP Logon Dialog Box

   b) Enter the transaction code **SU01** and in the *User* field, enter the ABAP user **AUTHSALES-##**, and choose the display option.

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

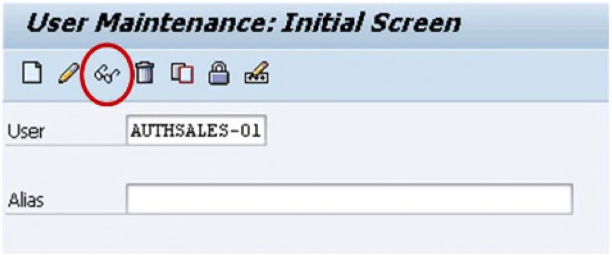


Figure 415: User Maintenance: Initial Screen

c) Select the *Profiles* tab and verify that the *HA900SALES* profile has been assigned to your user.



Figure 416: Profiles Tab

d) Double-click on the profile name to display a list of authorization objects.

e) Expand the object *V_VBAK_VKO* and view the fields that are part of that object.

f) Expand the field *VKORG* and observe the data value that the profile grants access to, in this case, 1000.

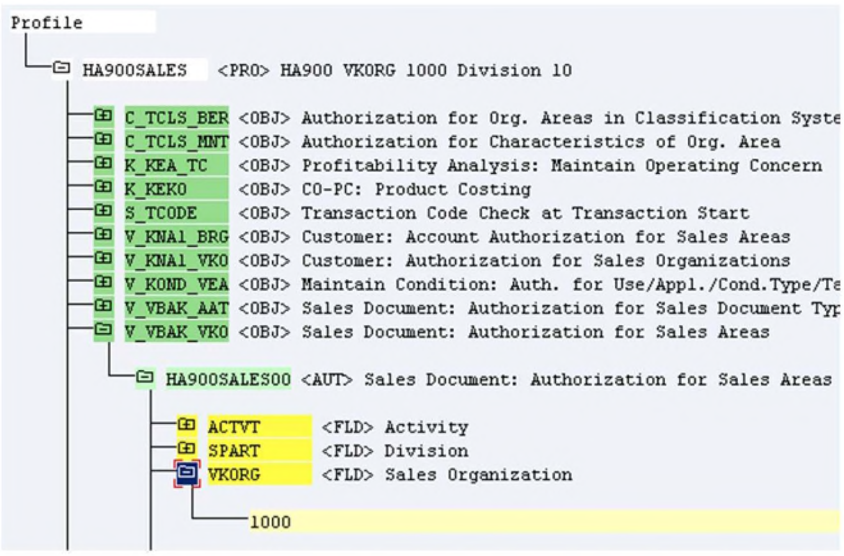


Figure 417: Profile Authorization Objects

5. Generate an SAP HANA analytic privilege from the ABAP user AUTHSALES-## and ensure that the new analytic privilege has been assigned the same Sales Organization value as the ABAP authorization.

   a) In the SAP HANA Studio, select the menu option *Analytics Authorization → Generate Analytic Privilege*.

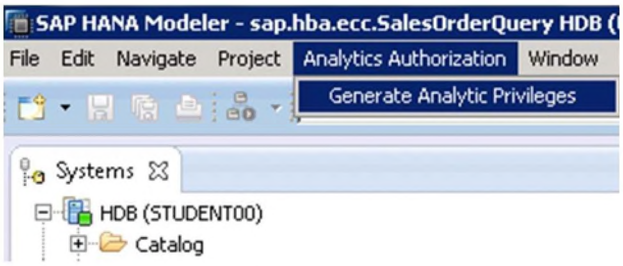


Figure 418: Analytics Authorization — Generate Analytic Privileges

b) Select the instance under user STUDENT## in order to begin. It can take a few moments for the dialog to appear.

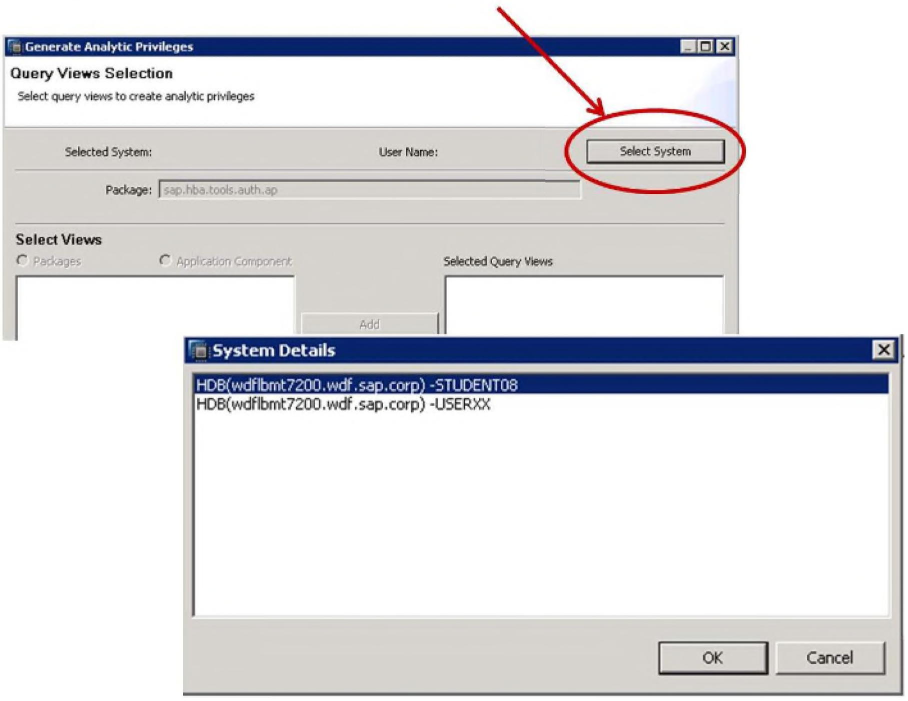


Figure 419: System Details

c) Expand the Packages hierarchy at *sap* → *hba* → *ecc*.

d) Highlight the view *SalesOrderQuery* and choose *Add* to move it to the *Selected Query Views* pane.

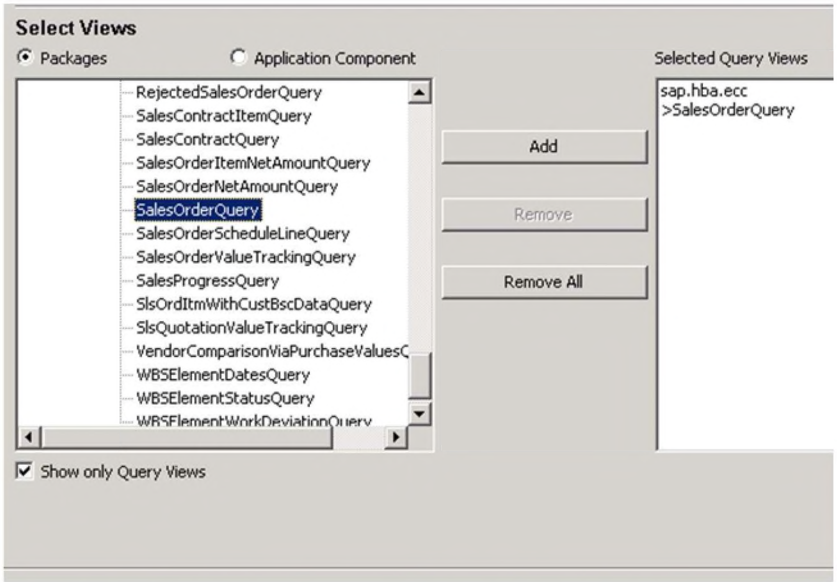


Figure 420: Moving SalesOrderQuery to the Selected Query Views

e) Choose *Next*.

f) In the *SAP Client* field, enter **800**.
The list of users that are created in the ABAP system under client 800 are now shown.

g) Find your ABAP user **AUTHSALES-##** (where ## is you number) and highlight it then choose *Add* to move it to the selection pane on the right.

> Hint:
> Use the search field to locate you user quickly, there are a lot of users in this list.

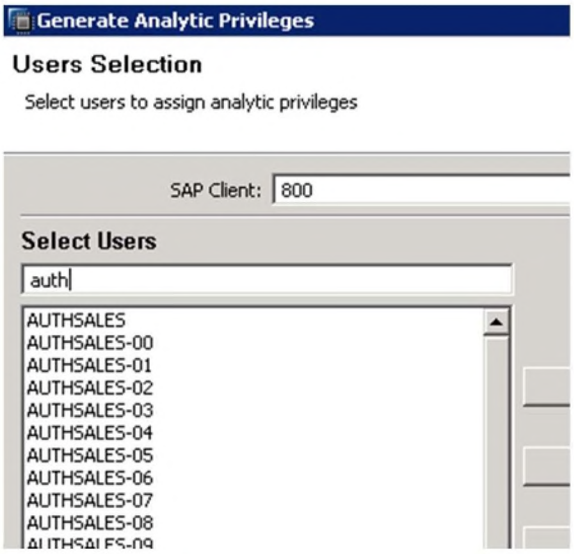


Figure 421: Users Selection

h) Choose *Finish*. In the job log, you should see the Analytic Privilege generation has completed successfully.



Figure 422: Finish Button

i) To find the generated Analytic Privilege, open the Analytic Privilege to observe the data values assigned to each attribute.

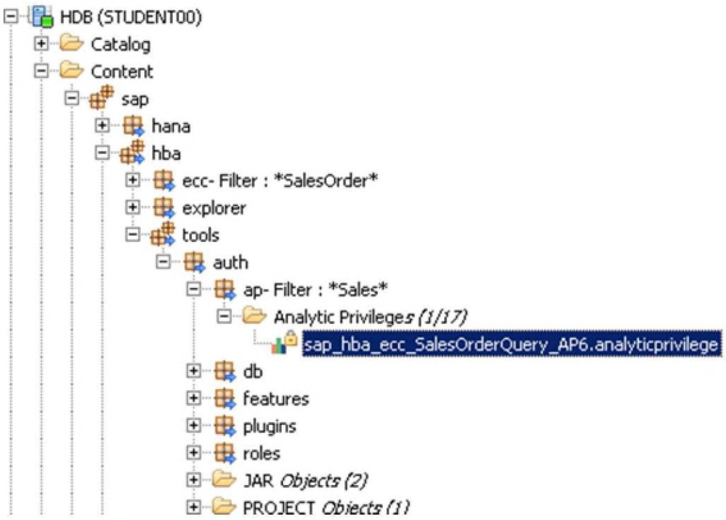


Figure 423: Analytic Privilege

6. View the analytics metadata supplied with the standard SAP HANA live view *SalesOrderQuery*.

   a) Open the *SalesOrderQuery* view, and verify that you can see the *Properties* view (*Window* → *Show View* → *Properties*).

   b) Select *Analytics MetaData* to observe the mapping of ABAP fields to HANA attributes.

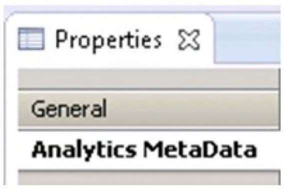


Figure 424: Properties View

   c) View the mappings in the properties of the view.



Figure 425: Mappings

7. Assign the newly generated Analytic Privilege to the SAP HANA user *USER##* then perform another data preview to ensure the user is able to view only Sales Organization *1000* for the view *SalesOrderQuery*.

a) Under the instance that uses the user *STUDENT##*, open the user *USER##*.

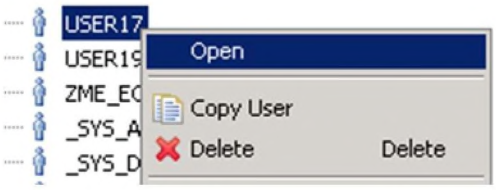


Figure 426: Open User##

b) Select the *Analytic Privileges* tab.

c) Choose +.

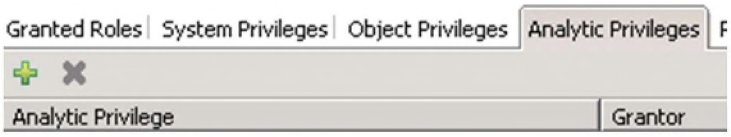


Figure 427: Analytic Privileges Tab

d) Highlight the privilege found and choose *OK*.

Choose *Deploy*.

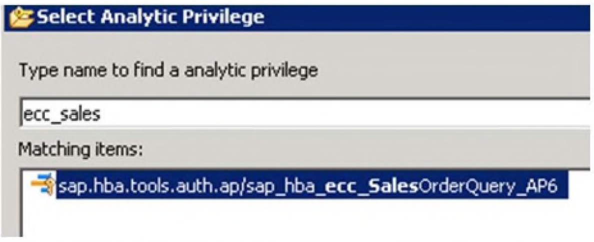


Figure 428: Select Analytic Privilege

e) Verify that user *USER##* can now view the data by expanding the hierarchy.

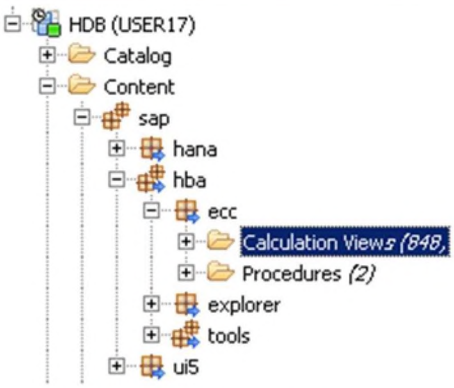


Figure 429: User## Calculation View

f) In the context menu, choose *Data Preview*.

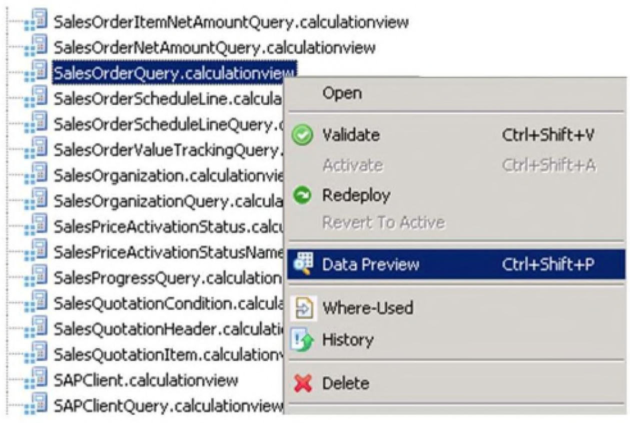


Figure 430: Data Preview Menu Option

g) At the prompt, in the *From* field, enter client **800**.

h) View the data. Notice that Sales Organization *1000* is the only value you are authorized to see.

SAPClient	SalesOrganization	SalesOrder
800	1000	0000011984
800	1000	0000011970
800	1000	0000011969
800	1000	0000011968
800	1000	0000011957
800	1000	0000011988
800	1000	0000011987
800	1000	0000011986
800	1000	0000011985
800	1000	0000011956
800	1000	0000011746
800	1000	0000011749
800	1000	0000011747

Figure 431: Sales Organization 1000

i) Select the *Distinct Values* tab and choose the *SaleOrganization* attribute. You should see there is only one distinct value *1000*.

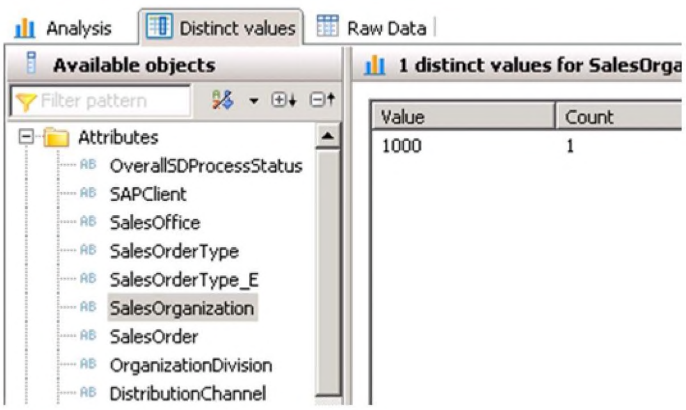


Figure 432: Distinct Values Tab

LESSON SUMMARY

You should now be able to:

- Generate SAP HANA Live Privileges

Unit 7
Lesson 4

Migrating Analytic Privileges

LESSON OVERVIEW

In this lesson, you will learn how to convert Classical Analytic Privilege into SQL Analytic Privileges with the *Migrate* utility available in the quick view.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Convert Classical Analytic Privileges to SQL-Based Analytic Privileges

Using the Migrate Utility

The migration tool provided with SAP HANA SPS11 can convert Classical Analytic Privileges into SQL-based Analytic Privileges.

The process is similar to the migration of Attribute or Analytic Views to Calculation Views.

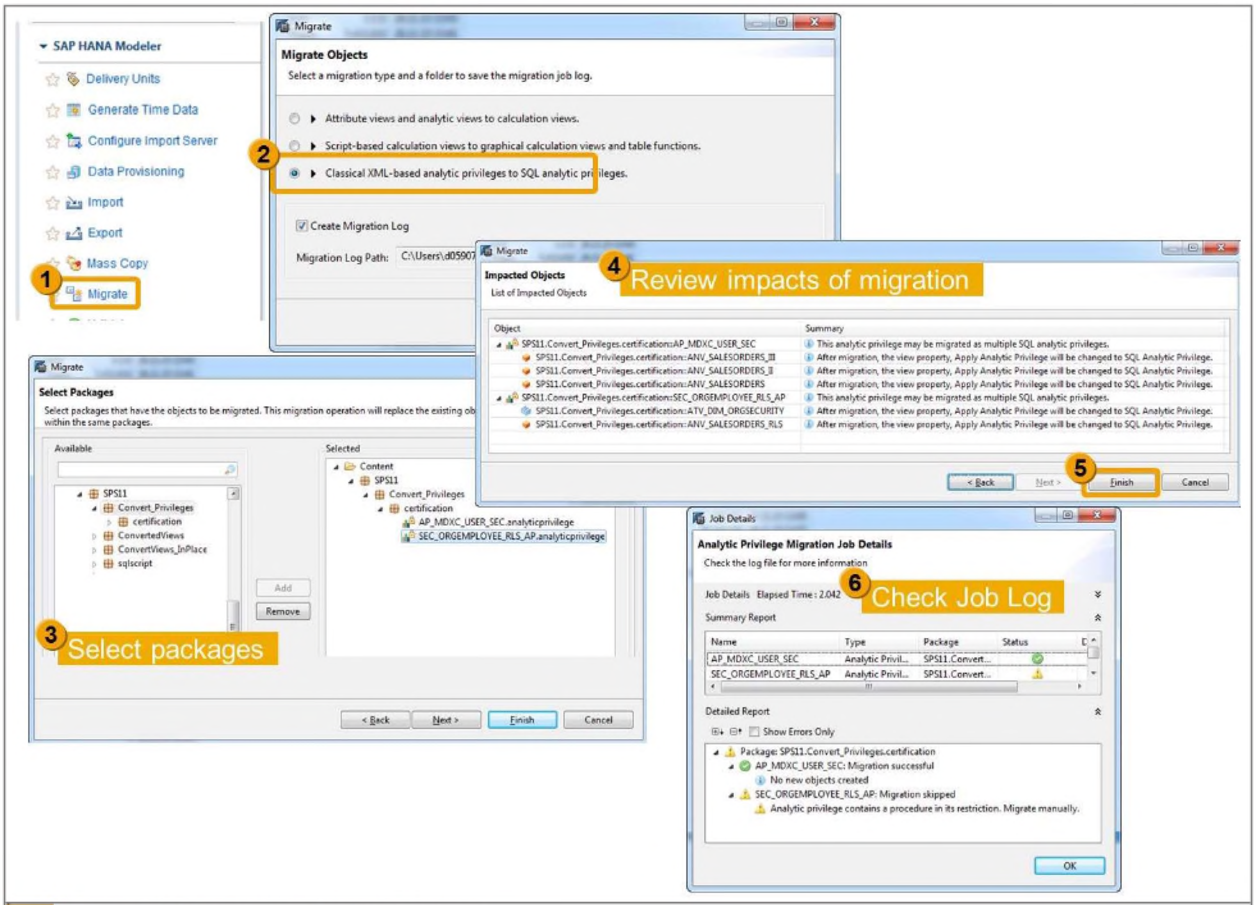


Figure 433: Migrating Analytic Privileges

Checking the List of Migrated Objects and Job Log

The list of impacted objects provides info on which information views will be adjusted automatically. The *Apply Privileges* property of Calculation Views will be changed from *Classical* to *SQL Analytic Privileges.*

> Note:
> This change is made automatically even if the packages containing the Calculation Views have not been selected for migration.

The list also mentions the information views that will not be migrated, and will need additional manual steps to complete the migration. Typical use cases include:

- When the source Analytic Privilege is designed to *Apply to all information models*.
- When the source Analytic Privilege is dynamic, that is, uses catalog or repository procedures to define the restrictions.

  Indeed, the way to implement dynamic Analytic Privileges differs between Classic ones and SQL ones.
- When one source Analytic Privilege is migrated to more than one SQL Analytic Privileges.

After migration, the job log provides, among others, information on the SQL Analytic Privileges that are not granted to any user or role. This helps you identify manual steps needed to complete the migration.

Like for other objects you migrate, Analytic Privileges can be migrated in several steps, to reach a consistent state before activating all the migrated objects.

Recommendations

As for the migration of other types of objects, you should apply the following recommendations:

Key Recommendations for the Migration Process

- Take a back-up of all objects before executing the migration
- Start migration with a clean workspace, with all Information Views activated
- Migrate Information Views before migrating Analytic Privileges

  If you plan to migrate both Attribute and Analytic Views to Calculation Views and Classical Analytic Privileges to SQL Analytic Privileges, the migration should be done in two distinct steps. Information Views first, then Analytic Privileges.
- Undoing migration changes is only possible before activating the objects

LESSON SUMMARY

You should now be able to:

- Convert Classical Analytic Privileges to SQL-Based Analytic Privileges

Learning Assessment

1. Which of the following are valid types of privileges in SAP HANA?

   Choose the correct answers.

   - [ ] A System privileges
   - [ ] B Attribute privileges
   - [ ] C Package privileges
   - [ ] D Application privileges
   - [ ] E Data Privileges

2. Which scenario is supported to base an Analytic Privilege on a hierarchy defined on an attribute column?

   Choose the correct answer.

   - [ ] A Create a Classical Analytic Privilege based on a level hierarchy
   - [ ] B Create a Classical Analytic Privilege based on a parent-child hierarchy
   - [ ] C Create an SQL Analytic Privilege based on a level hierarchy
   - [ ] D Create an SQL Analytic Privilege based on a parent-child hierarchy

3. What are the features of Dynamic analytic privileges?

   Choose the correct answers.

   - [ ] A Reuse the same analytic privilege for several users who need to access different data
   - [ ] B Filter the data based on a variable entered by the user when they query the view
   - [ ] C Change the filter condition in the underlying tables and views

4. When you generate Analytic Privileges in SAP HANA from an ABAP System with the SAP HANA Live Authorizations Assistant, the SAP HANA users are automatically assigned to the generated roles.

   Determine whether this statement is true or false.

   - [ ] True
   - [ ] False

5. It is recommended to migrate the Attribute and Analytic Views to Calculation Views before migrating the Analytic Privileges that secure them.

   Determine whether this statement is true or false.

   - [ ] True
   - [ ] False

Unit 7

Learning Assessment - Answers

1. Which of the following are valid types of privileges in SAP HANA?

   Choose the correct answers.

   - [X] A System privileges
   - [ ] B Attribute privileges
   - [X] C Package privileges
   - [X] D Application privileges
   - [ ] E Data Privileges

2. Which scenario is supported to base an Analytic Privilege on a hierarchy defined on an attribute column?

   Choose the correct answer.

   - [ ] A Create a Classical Analytic Privilege based on a level hierarchy
   - [ ] B Create a Classical Analytic Privilege based on a parent-child hierarchy
   - [ ] C Create an SQL Analytic Privilege based on a level hierarchy
   - [X] D Create an SQL Analytic Privilege based on a parent-child hierarchy

3. What are the features of Dynamic analytic privileges?

   Choose the correct answers.

   - [X] A Reuse the same analytic privilege for several users who need to access different data
   - [ ] B Filter the data based on a variable entered by the user when they query the view
   - [X] C Change the filter condition in the underlying tables and views

4. When you generate Analytic Privileges in SAP HANA from an ABAP System with the SAP HANA Live Authorizations Assistant, the SAP HANA users are automatically assigned to the generated roles.

   Determine whether this statement is true or false.

   - [ ] True
   - [X] False

5. It is recommended to migrate the Attribute and Analytic Views to Calculation Views before migrating the Analytic Privileges that secure them.

   Determine whether this statement is true or false.

   - [X] True
   - [ ] False

UNIT 8 Optimization of Models

UNIT OBJECTIVES

- Apply good modeling approaches to SAP HANA
- Understanding the cost of SQL Statements

Unit 8
Lesson 1

Applying Good Modeling Approaches

LESSON OVERVIEW

There are many choices to make when developing SAP HANA models. making the wrong choice can severely affect the function and performance of your model. In this lesson, we will guide you towards good modeling approaches.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Apply good modeling approaches to SAP HANA

Different Types of Engines for Data Processing

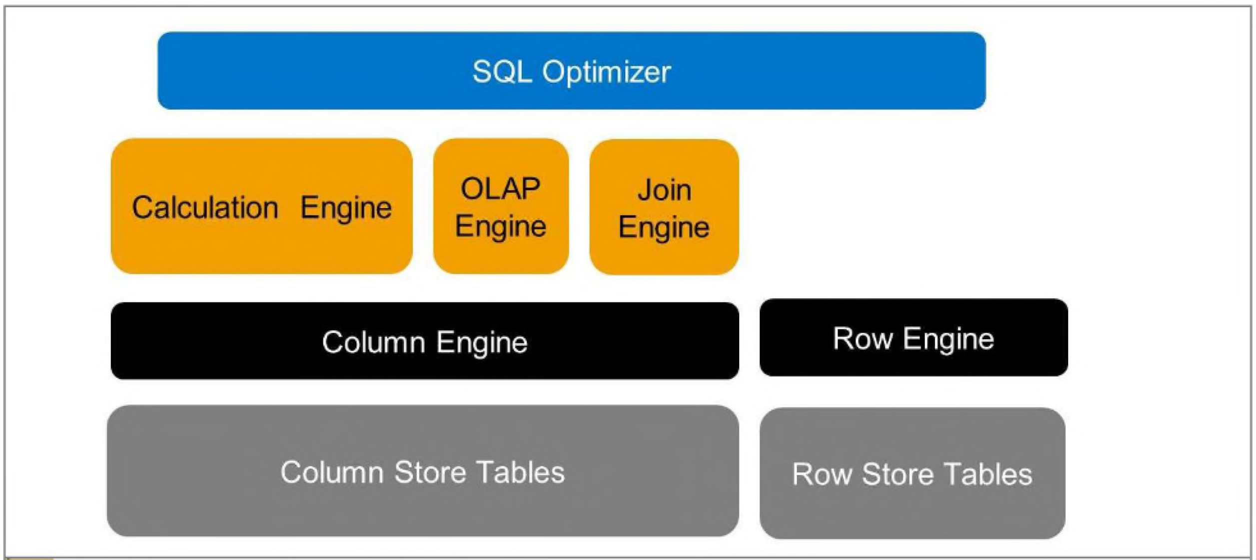


Figure 434: SAP HANA Engine Overview

The SAP HANA architecture provides different types of engines:

- Join Engine

  The Join Engine is used to process joins between column tables (not row tables), and also take care of other basic functions such as filtering, aggregations, and sorting.

- OLAP Engine

  The OLAP Engine is used for calculation and dimensional aggregations based on a star schema. It is called from analytic views and from calculation views that use the star join.

- Calculation Engine

  The Calculation Engine is used on top of the OLAP engine and the Join Engine for complex calculations that cannot be processed by the Join Engine or OLAP Engine.

> Note:
> For spatial and graph queries, there are also specific engines used.

SAP HANA Engine Overview II

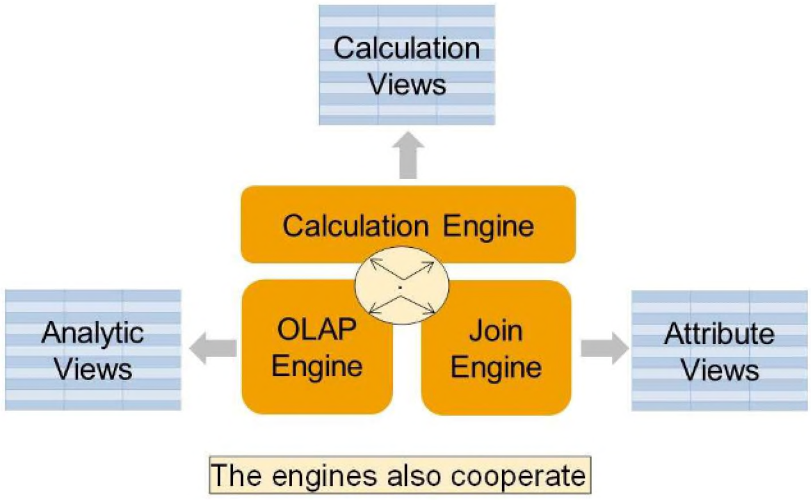


Figure 435: SAP HANA Engine Overview II

When building new information models, it is useful to keep in mind which engine is predominantly used by which type of model, as described in a simple way in the figure, SAP HANA Engine Overview II. There is quite a lot of overlap in the capabilities of the engines, and to identify which engine(s) is (are) actually called, you must use the Explain Plan tool (covered later).

A key message is that the overall SAP HANA query optimization process can hand over each type of data processing to the engine that will perform the best, depending on the involved models and queries sent to SAP HANA.

> Note:
> For example, the Dimension calculation views are delegated to the Join Engine whenever possible, and the Cube with Star Join to the OLAP Engine.

Here are a couple of examples of common exceptions to this default assignment of engines by type of view:

- The execution of a Dimension View with calculated attributes, or a Cube with Star Join View and calculated measures, generally involves the Calculation Engine, and is not processed exclusively by the Join Engine and the OLAP Engine.
- An attribute view or Dimension Calculation View with just one table does not involve the Join Engine.

Best Practices for Data Modeling

When you build any information model, you must keep in mind a number of principles that will help you to build high-performance models.

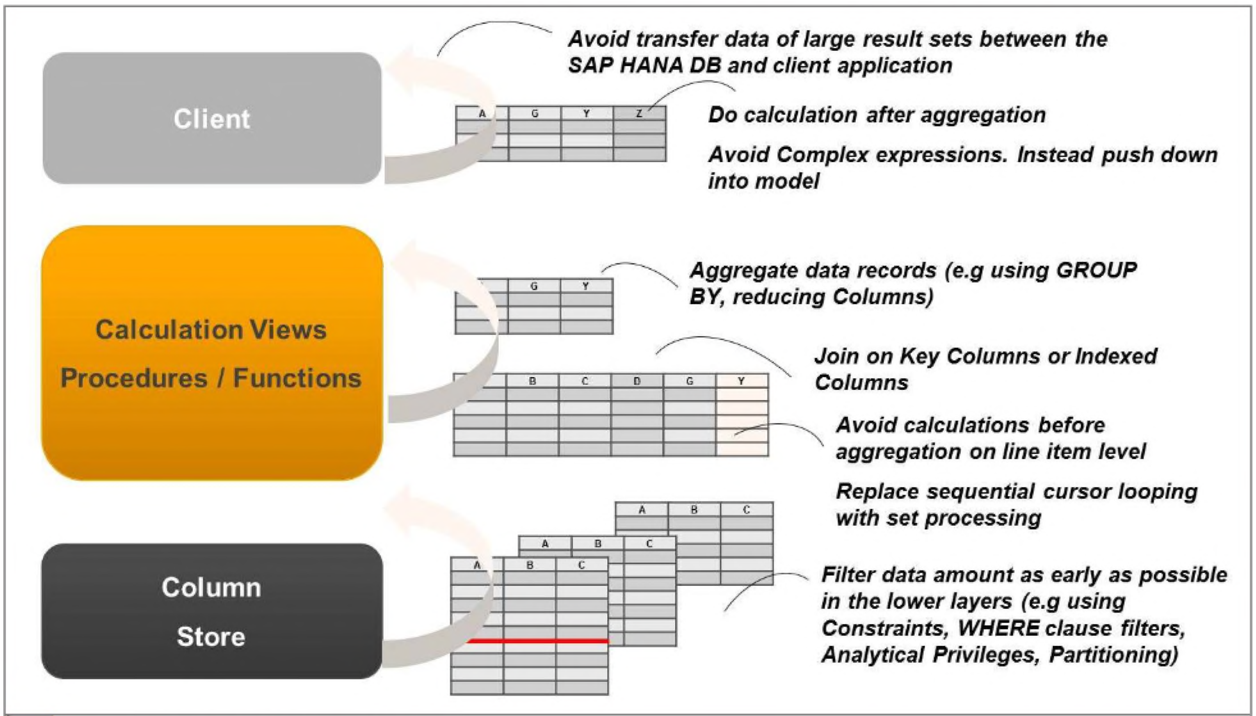


Figure 436: General Modeling Principles

Filter and Aggregation Push Down

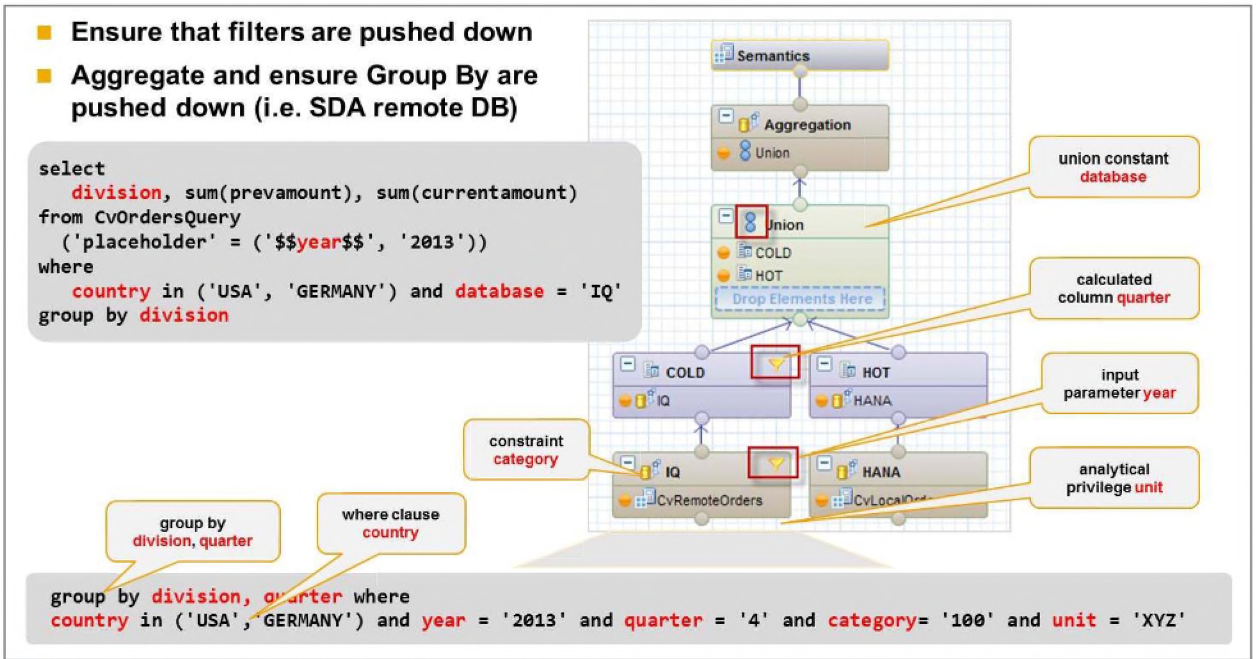


Figure 437: Filter and Aggregation Push Down

It is recommended to perform filtering and aggregation as early as possible in the calculation scenario.

Dedicated List of Values Views/Tables for Variables and Input Parameters

- **Faster value help dialogs**
- **Consistent Lists of Values across consuming views**
- **Dependencies between value help views**
- **Support for Analytic Privileges**

```
select
   teamname, playername, rank,
   sum(playergoals), sum(teamgoals)
from
CvSoccerPlayers(
  'placeholder'=('$$top_n_players$$','2'),
  'placeholder'=('$$ip_team$$','paderbom')
)
group by teamname, playername, rank
order by teamname
```

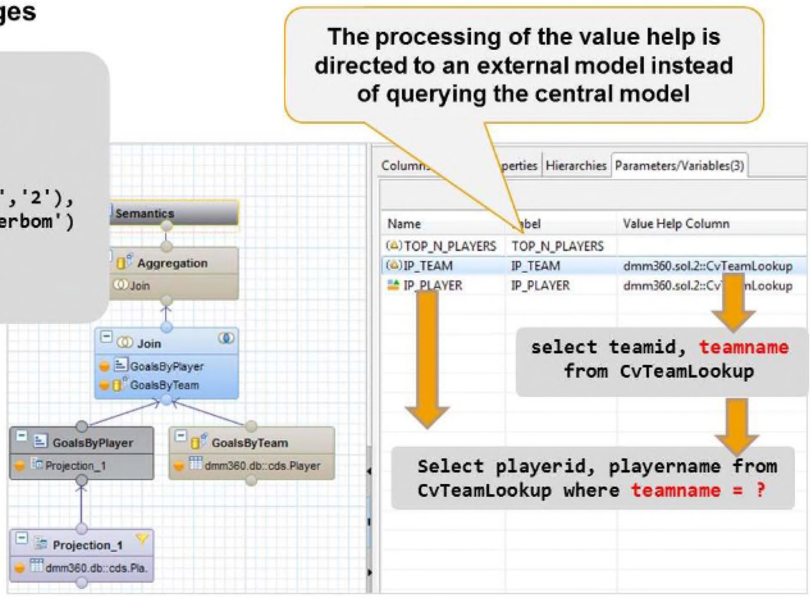


Figure 438: Using Dedicated Value Help Views or Tables

For performance issues, we generally recommend that you use dedicated Views/Tables to generate the lists of values for variables and inputs with the *Column* parameter type.

This enables a faster display of the list of values, without reading the entire view or table referenced by the Calculation View.

It is also a best practice to ensure that the list of values is consistent from one view to another.

SQL Execution of Calculation Views

When the *Execute In* property is switched to *SQL Engine* in a graphical calculation model, the Calculation Engine tries to generate SQL from each node during runtime.

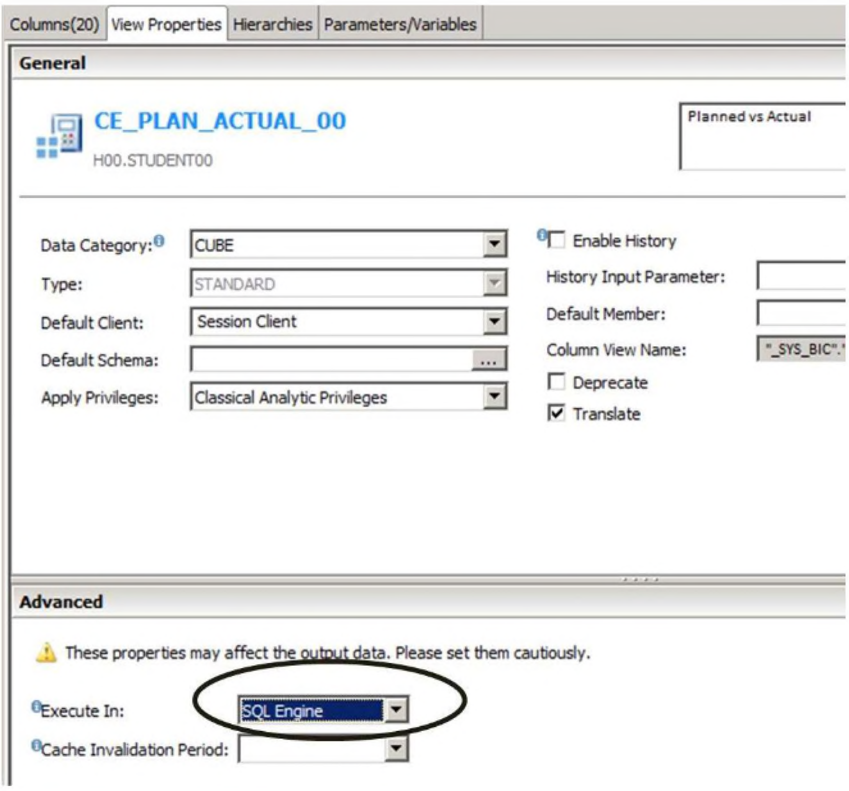


Figure 439: Execute in SQL Engine

After generating SQL for each node (if possible), the SQL statements are combined into one big SQL statement which is then passed to SQL interface. This might be beneficial as the SQL interface also involves the standard SQL optimizer, which implements a lot more optimization rules than Calculation Engine. For example, rules that determine the optimal join sequence.

Caution:
This option, introduced with SAP HANA revision 62, was especially useful before SAP HANA SPS09. With the SPS09 release, SAP HANA always tries to leverage additional SQL optimizations.

For more information, see SAP Note 2223597 – Implicit SQL optimization of SAP HANA Calculation Views.

Rules for Optimal SQL Generation

The *Execute in: SQL Engine* option, and the corresponding (implicit) SQL optimization introduced with SAP HANA SPS09, are only efficient if every node can be translated so one big SQL statement can be generated.

To gain optimal SQL generation, the following rules should be followed:

- Do not include any source other than tables or Graphical Calculation Views. If the Calculation View is included, all of these rules must also hold for the included calculation view.

  Explicitly, do not add any Analytic View, Attribute View, or Scripted Calculation View, this might affect optimization.

- For operations, only use projection, union, join, or aggregation.

Do not use special column store functions in filter or calculated attribute expressions as they might not be convertible. For instance, the functions date() and time() only work on the column store data types and therefore cannot be converted.

- Until SPS10, SAP HANA could differently handle Column Engine and SQL expression languages. In particular, you had to use specific operators to ensure a harmonized execution of operations involving NULL values. For example:`1 + NULL` did not provide the same result depending on the engine.

  SAP HANA SPS11 now offers an implicit conversion of such expressions, so that they behave identically in SQL and Column Store, and do not generate result differences depending on the engine that processes the calculation.

  For example, the expression `1 + NULL` in the Column Store will be implicitly converted to `ADD_NAZ(1,NULL)`, and then return **1**, regardless of the execution mode.

- If a currency/unit conversion or counter measure needs to be defined it has to be defined in the output node of the main scenario.

  If a currency conversion or counter measure is modeled into an included scenario, SQL optimization will be broken. (as these special measures can be handled more efficiently in Calculation Engine).

> Note:
> As an example of optimal design, the SAP HANA Live views follow the rules above, so these views can be fully optimized using only the SQL engine.

Best Practices When Writing SQL

One of the benefits of using graphical modeling tools is that you don't have to concern yourself with writing SQL code. However, there are many times when you do need to write the SQL code directly. For example, when you are developing a scripted calculation view, or a procedure.

Whilst writing the SQL directly offers the most flexibility to control exactly what you want to achieve and how you want to achieve it, this does mean that it is easy to write code that may not be optimal for SAP HANA. Even experienced SQL coders should pay attention to these guidelines to avoid applying techniques that do not work well with SAP HANA.

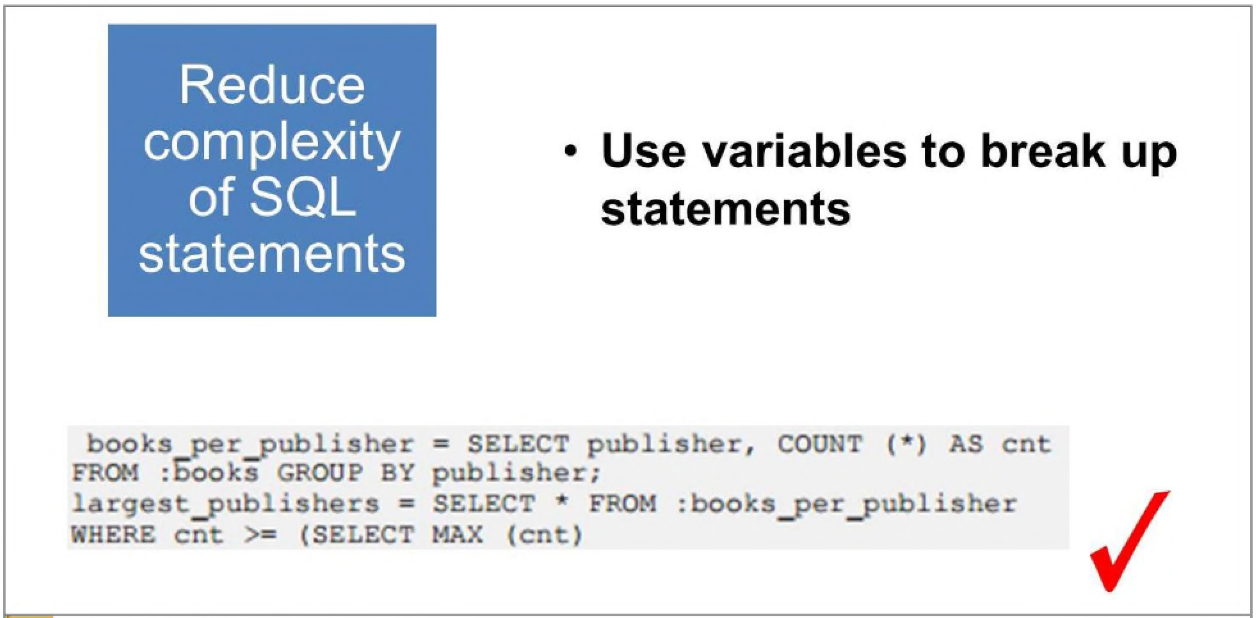


Figure 440: SQL Best Practices 1

Reduce complexity of SQL statements by using variables. Variables are used to capture the results of each SQL statement and can then be used as the inputs for subsequent steps. This makes understanding the code much easier and does not sacrifice performance.

Apply Multi-level Aggregation

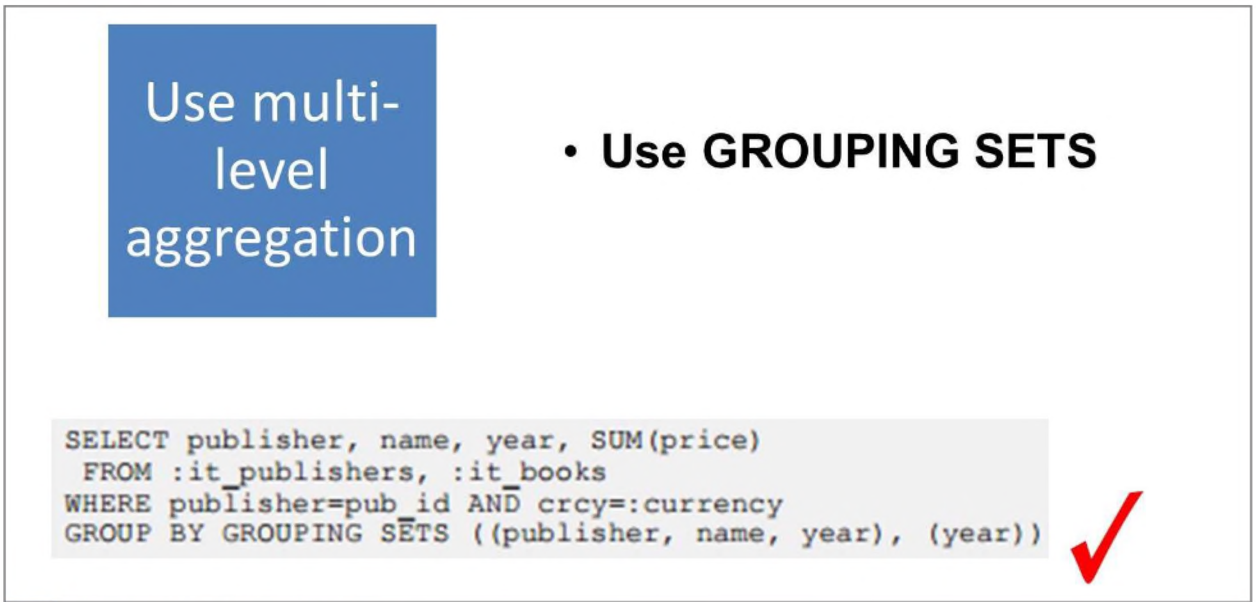


Figure 441: SQL Best Practices 2

Apply multi-level aggregation by using grouping sets. Grouping sets aggregate source data at multiple levels all from only one SQL statement. This means you only read the data once but produce multiple aggregation results that subsequent SQL steps can read.

Reduce Dependencies

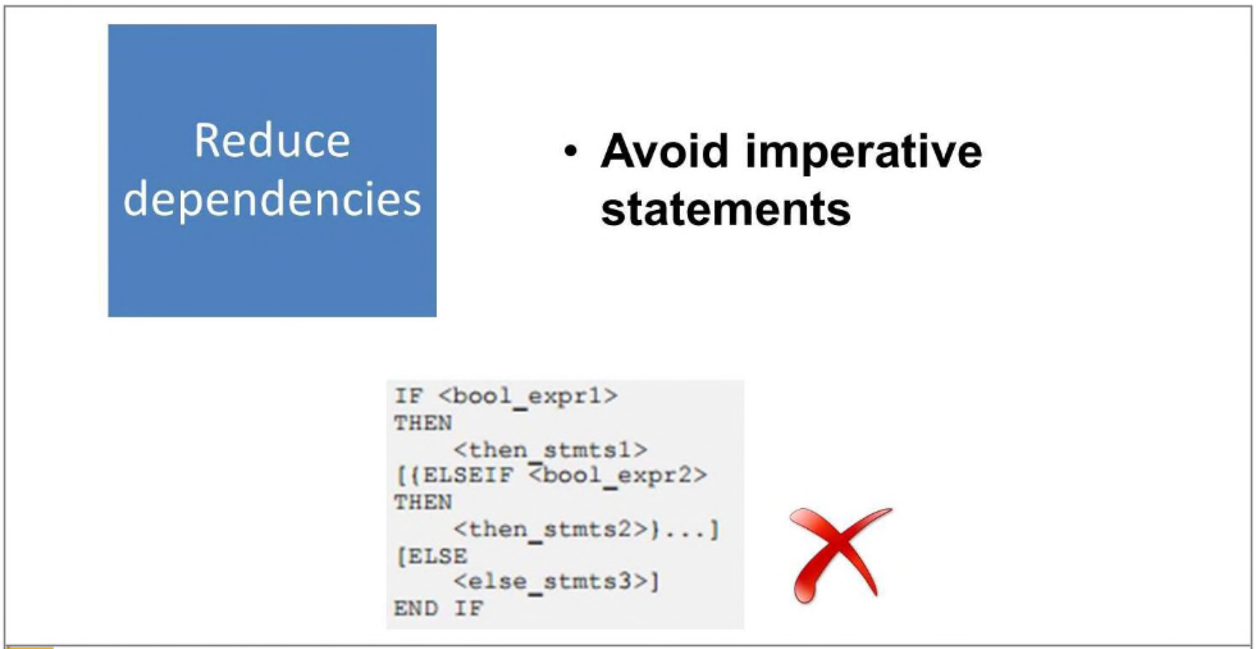


Figure 442: SQL Best Practices 3

Reduce dependencies by ensuring your SQL steps are independent from each other. The reason for this is to provide the best possible opportunity for parallelization and therefore obtain the best performance.

The best way to reduce dependencies is to avoid imperative statements (conditions, loops, case) and try to stick with declarative statements (select, CE functions). Of course, sometimes, it is not possible to avoid writing imperative statements.

Avoid Mixing CE Plan Operators and SQL Queries

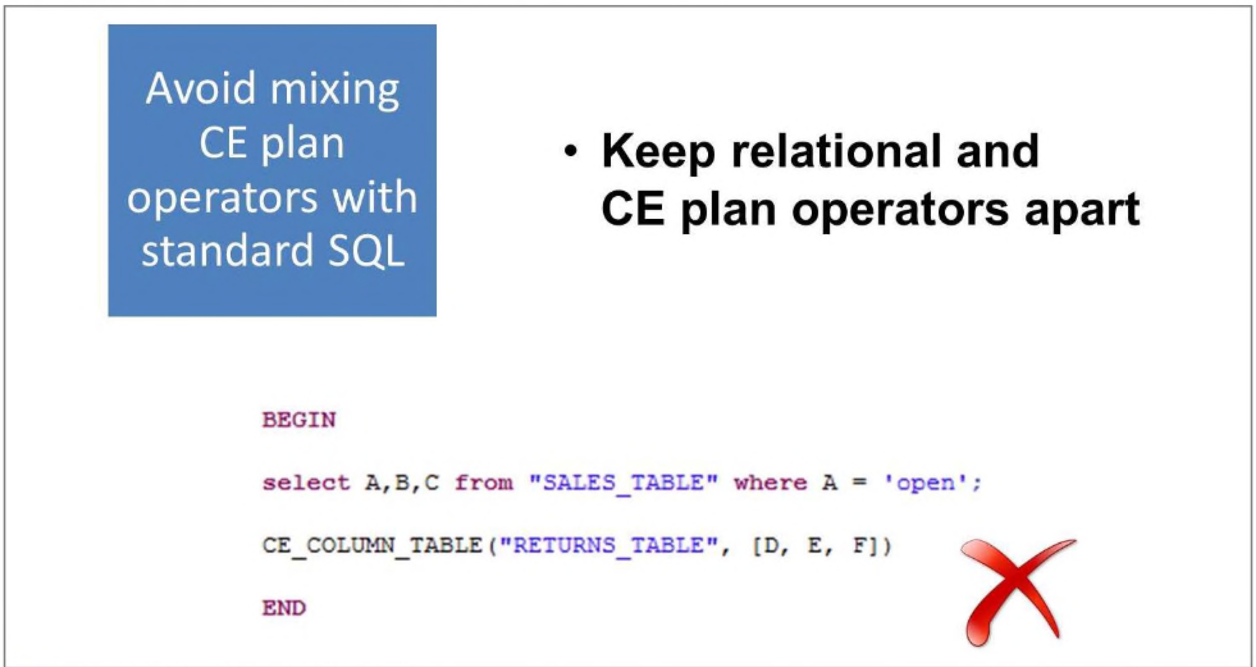


Figure 443: SQL Best Practices 4

Avoid mixing CE plan operators and SQL queries by ensuring that you do not create SQL code that calls the SQL engine and the calculation engine. Each engine uses a different optimization

method and mixed code, which requires different engines that cannot be fully optimized. Use code that calls only one of the engines.

> Note:
> SAP recommends that you use SQL rather than Calculation Engine Plan Operators with SQLScript. The execution of Calculation Engine Plan Operators is bound to processing only within the calculation engine and does not allow a possibility to use alternative execution engines, such as the SQL engine.
>
> As most Calculation Engine Plan Operators are converted internally and treated as SQL operations, the conversion requires multiple layers of optimizations. This can be avoided by direct SQL use. Depending on your system configuration and the version you use, mixing Calculation Engine Plan Operators and SQL can lead to significant performance penalties when compared to plain SQL implementation.

> Note:
> You can learn more about SAP HANA Optimization with the course, HA215 – Performance Tuning and Optimization. Go tohttp://training.sap.com.

LESSON SUMMARY

You should now be able to:

- Apply good modeling approaches to SAP HANA

Unit 8
Lesson 2

Understanding the Cost of SQL Statements

LESSON OVERVIEW

In this lesson, we will highlight some common mistakes that are often made when writing SQL code that will run on SAP HANA.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Understanding the cost of SQL Statements

Understanding the cost of SQL statements

In SAP HANA, to identify SQL queries that are inefficiently processed, you can technically and logically analyze the steps SAP HANA took to process those queries.

From a technical perspective, analyzing query plans allows you to identify long running steps, understand how much data is processed by the operators, and see whether data is processed in parallel. However, if you understand the idea and purpose behind the query, you can also analyze query plans from a logical perspective and consider questions such as:

- Does SAP HANA read data from multiple tables when only one is required?
- Does SAP HANA read all records when only one is required?
- Does SAP HANA start with one table even though another table has a much smaller result set?

SAP HANA studio provides the following tools for query plan analysis:

- Plan Explanation
- Plan Visualization

Explain Plan

You can generate a plan explanation for any SQL statement in the SQL console of the SAP HANA Studio. You can use this to evaluate the execution plan that the SAP HANA database will follow to execute an SQL statement.

	OPERATOR_NAME	OPERATOR_DETAILS	EXECUTION_ENGINE
1	COLUMN SEARCH	PERIOD, SALESORG, PLANACTUAL, MANDT, KUNNR, LAND1, O...	OLAP
2	AGGREGATION	GROUPING: PERIOD, SALESORG, PLANACTUAL, MANDT, KUNN...	OLAP
3	COLUMN SEARCH	[FACT] PERIOD, SALESORG, PLANACTUAL, MANDT, KUNNR, LA...	COLUMN
4	COLUMN UNION ALL	(STUDENT00/CEA1_00.PERIOD,STUDENT00/CEP1_00.PERIOD)...	COLUMN
5	COLUMN SEARCH	STUDENT00/CEA1_00.MANDT, STUDENT00/CEA1_00.KUNNR, S...	COLUMN
6	AGGREGATION	GROUPING: STUDENT00/CEA1_00.PERIOD, STUDENT00/CEA1_...	COLUMN
7	COLUMN VIEW		COLUMN
8	COLUMN SEARCH	STUDENT00/CEP1_00.MANDT, STUDENT00/CEP1_00.KUNNR, S...	COLUMN
9	AGGREGATION	GROUPING: STUDENT00/CEP1_00.PERIOD, STUDENT00/CEP1_...	COLUMN
10	COLUMN VIEW		COLUMN

Figure 444: Explain Plan

One of the most useful pieces of information this tool provides is to identify the engines used in the execution. During the design-time modeling phase it is not possible to identify precisely which execution engines will be called. This depends very much on how your model is built and which functions you are using. Remember, you must try to avoid calling multiple engines, unless absolutely necessary, to maintain good performance. If multiple engines are called then all data is passed between them each time, which can lead to sub-optimal performance. A skilled modeler using good modeling practices will avoid unnecessary calling of execution engines.

The *Explain Plan* tool provides the essential visibility of the execution engines and also reveals how many times and in which order they are called.

The *Explain Plan* tool is accessed from the *SQL console*. Once your SQL code is visible, choose the option *Explain Plan* from the context menu.

Note:
From SAP HANA SPS10, it is possible to automatically generate a `Select` SQL statement from the design time models of any information view. Simply use the context menu from any model under the package node and choose *Generate Select SQL*.

This means that you do not have to locate the run-time object under `_SYS_BIC`, drag in into the SQL console, and then write the `Select` statement any longer.

Plan Visualize

To help you understand and analyze the detail of an execution plan of an SQL statement, you can generate a graphical view of the plan.

The *Plan Visualize* tool can provide the modeler with valuable information that can help them develop the most optimal models for best performance.

There are two modes for this tool *Prepare* and *Execute*.

Prepare produces a data flow with an estimated run time for all steps.

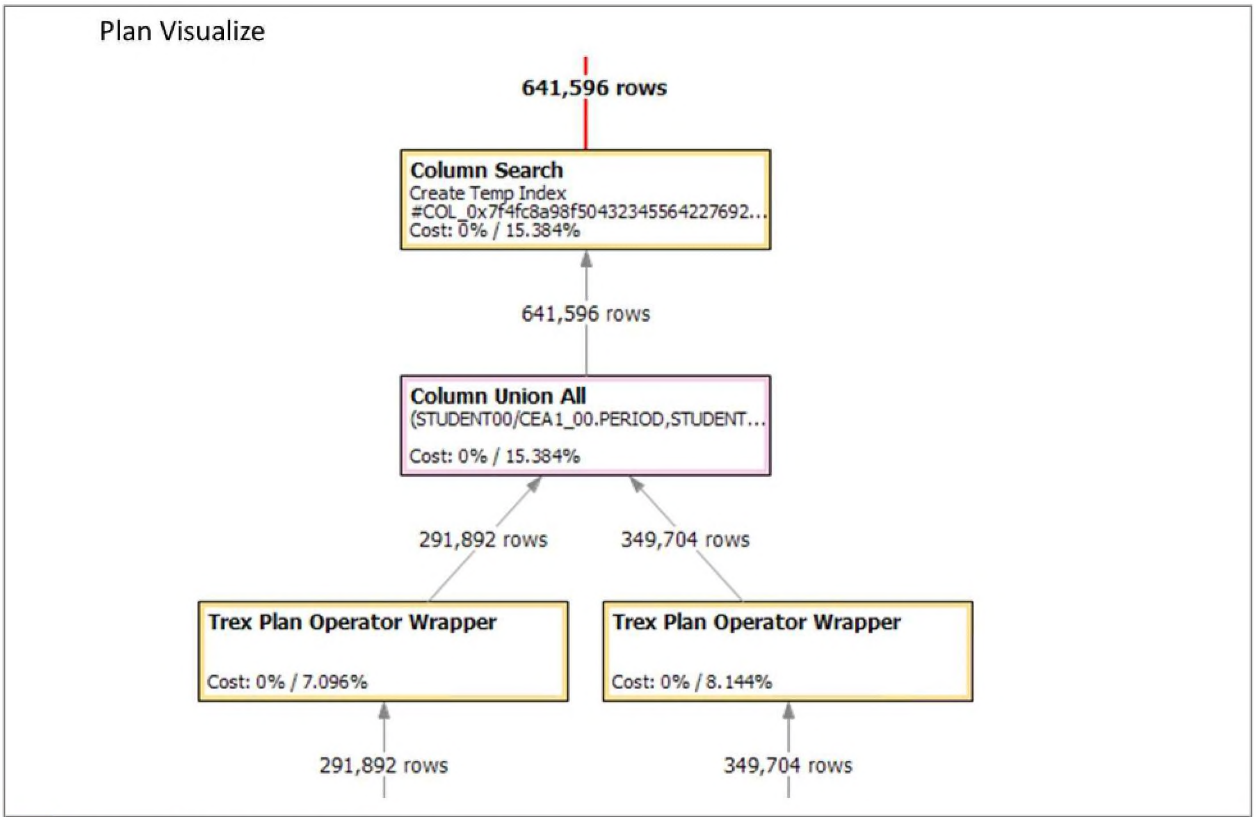


Figure 445: Plan Visualize Flow — Prepare phase

The *Plan Visualize* → *Execute* executes the SQL and provides actual run time values, not estimates. The results are presented across two tabs, *Overview* and *Executed Plan*.

> Note:
> Prior to SPS10 the execute plan option would only be possible once the prepare plan was chosen. With SPS10 you can now skip the prepare phase and go straight to execute.

Plan Visualize Execute — Overview tab

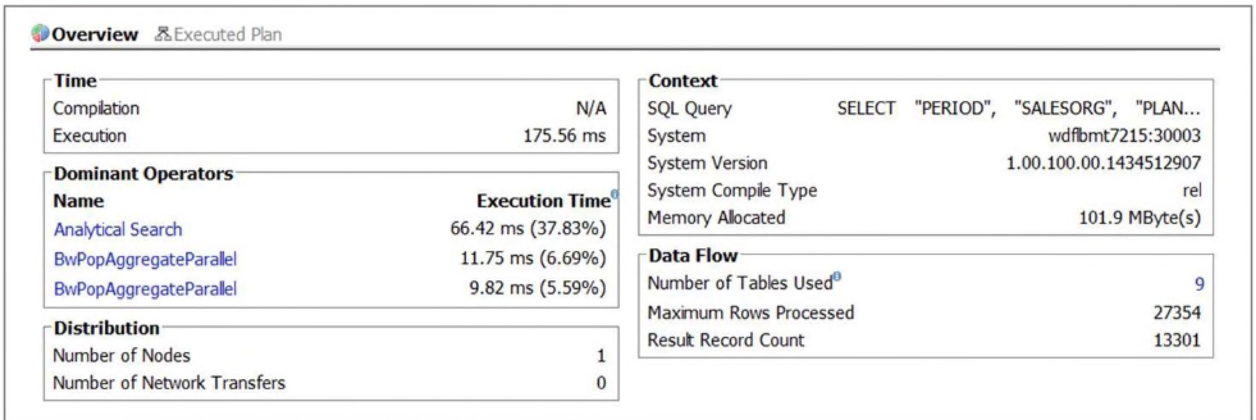


Figure 446: Plan Visualize Execute — Overview tab

There are many pieces of information that are provided on the *Overview* tab, here are some of the most useful for modelers:

- Total execution time

- Top 3 longest running steps (plus you can drill each for more details)
- How many tables were accessed
- How many records were read from each table (drill down)
- How many records are produced in the result

Plan Visualize Execute — Executed Plan tab

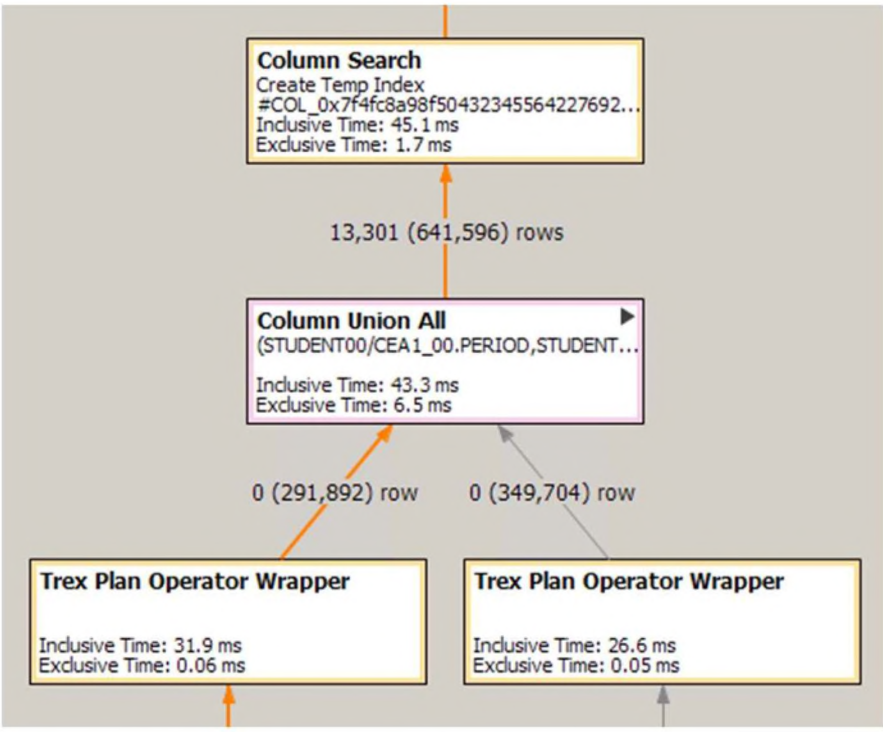


Figure 447: Plan Visualize Execute — Executed Plan tab

There is a huge amount of information provided on the *Executed Plan* tab. Here are some of the most useful for modelers:

- The sequence of steps in the flow
- Time taken on each processing step
- Cumulative time taken to reach each step
- How many rows are passed between steps
- Which operator is called at each step

You are able to drill down in the flow to obtain very detailed information so that you can examine what is happening at the most granular level of processing. Don't forget to use the zoom-in and zoom-out tools to make the display more comfortable as the screen quickly fills with information.

The *Visualize Plan* tool is accessed from the *SQL console*. Once your SQL code is visible in the SQL Console, choose *Visualize Plan* from the context menu and then decide if you wish to display a plan for preparation or execution at the lower menu option.

Executions times are always shown as inclusive and exclusive. Exclusive time is the time is has taken to execute only the step with no regard for any other preceding steps. Inclusive time is the total time is has taken to execute the step plus the other times for all preceding steps.

Timeline View

One of the most useful views to help you easily understand where most time is spent in the execution of your SQL is the *Timeline* view.

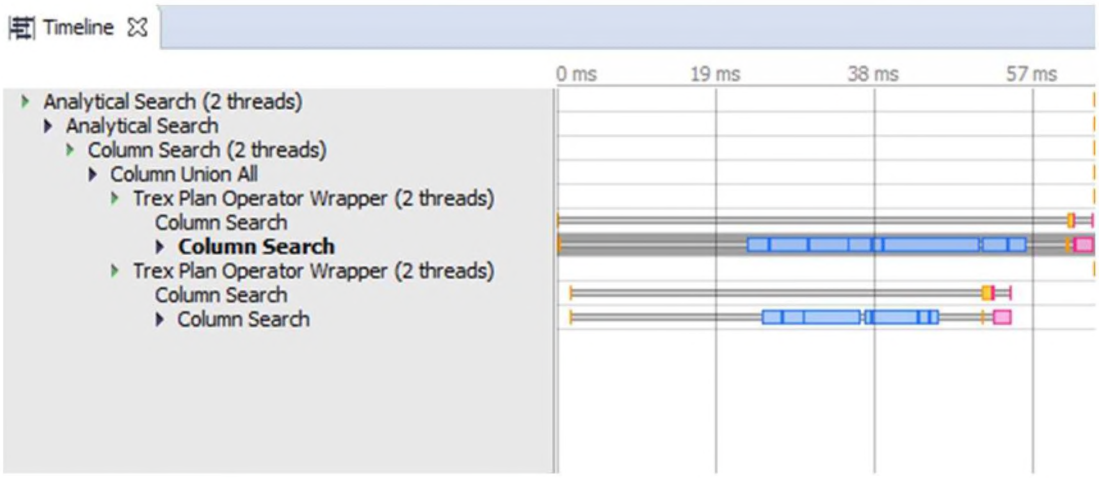


Figure 448: Timeline View

The *Timeline* view provides a complete overview of the execution plan based on the visualization of sequential time-stamps. The following pieces of information are available in the view:

- X-axis: time elapsed since query execution,
- Y-axis: list of operators
- Duration of each operator execution represented by rectangular bars
- The nature of each time-stamp (for example, open, fetch, close, and so on.)

Other Views for Analysis of SQL runtime

There are many other views that can provide additional information to help you understand all aspects of the execution behavior of your SQL These include the following:

- Operator list: lists detailed characteristics of all operators within a current plan this is also a great way to easily identify long running steps within a flow.
- Network view: visualizes sequential network data transfers between different servers based on their given timestamps when an execution plan is based on a distributed query.
- Tables used: cross: checks your business understanding about which tables are needed to fulfill a given SQL statement.
- Performance trace: displays the list of operators that have performance trace data. The data can be viewed by selecting any of the operators shown in the list.

Unit 8
Exercise 33

Use Explain Plan

Business Example

You want to optimize some of your script-based information models, and need some key information to help you understand how your modeled information view behaves at execution time.

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.

1. Using the *Explain Plan* tool, display the results for the *CE_PLAN_ACTUAL_00* calculation view, found under the *STUDENT00* package.

2. Which execution engines are used?

3. How many times is the *AGGREGATION* operator called in the plan?

4. Why is the *AGGREGATION* operator called multiple times for this model?

Unit 8
Solution 33

Use Explain Plan

Business Example

You want to optimize some of your script-based information models, and need some key information to help you understand how your modeled information view behaves at execution time.

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.

1. Using the *Explain Plan* tool, display the results for the *CE_PLAN_ACTUAL_00* calculation view, found under the *STUDENT00* package.
   a) Locate the *CE_PLAN_ACTUAL_00* calculation view using the path *Content → STUDENT00 → Calculation Views*.
   b) From the context menu of the *CE_PLAN_ACTUAL_00* calculation view, choose *Generate Select SQL*.
   c) From the context menu of the SQL console, choose *Explain Plan*.
2. Which execution engines are used?

   The column engine (COLUMN) and calculation engine (CALC) are used.
3. How many times is the *AGGREGATION* operator called in the plan?

   It is used three times.
4. Why is the *AGGREGATION* operator called multiple times for this model?

   Open this model and you will see that initially two analytic views are called. Each one of these analytic views would aggregate their measures over all their dimensions. Then, the results of these aggregations are combined in a union and finally all measures from the union would be aggregated by all the attributes in the aggregate node. This would mean we have three aggregate operations in total.

Unit 8
Exercise 34

Use Visualize Plan

Business Example

You want to find detailed information to help you understand more specifically how your modeled information view behaves at execution time.

1. Using the *Visualize Plan* tool, display the estimated execution plan for the *CE_PLAN_ACTUAL_00* calculation view, found under the *STUDENT00* package.

2. What percentage of the overall time/cost is estimated for the union step?

3. Using the *Visualize Plan* tool, display the actual execution plan for the *CE_PLAN_ACTUAL_00* calculation view, found under the *STUDENT00* package.

4. What is the total execution time?

5. What is the actual time it has taken to execute the plan up to, and including, the union step?

6. How many rows were generated by the union step?

Unit 8
Solution 34

Use Visualize Plan

Business Example

You want to find detailed information to help you understand more specifically how your modeled information view behaves at execution time.

1. Using the *Visualize Plan* tool, display the estimated execution plan for the *CE_PLAN_ACTUAL_00* calculation view, found under the *STUDENT00* package.
   a) Locate the *CE_PLAN_ACTUAL_00* calculation view using the path *Content → STUDENT00 → Calculation Views*.
   b) From the context menu of the *CE_PLAN_ACTUAL_00* calculation view, choose *Generate Select SQL*.
   c) From the context menu of the SQL console, choose *Visualize Plan → Prepare*.
2. What percentage of the overall time/cost is estimated for the union step?

   About 12%
3. Using the *Visualize Plan* tool, display the actual execution plan for the *CE_PLAN_ACTUAL_00* calculation view, found under the *STUDENT00* package.
   a) Return to the SQL Console tab with the SELECT statement generated on top of the *CE_PLAN_ACTUAL_00* calculation view.
   b) From the context menu of the SQL console, choose *Visualize Plan → Execute*.
4. What is the total execution time?

   The actual execution time will vary and is dependent on the current load of the SAP HANA system at the time you run the plan. Choose the *Overview* tab, and you will see an execution time somewhere in the range 150 – 200 ms.
5. What is the actual time it has taken to execute the plan up to, and including, the union step?

   The actual execution time will vary and is dependent on the current load of the SAP HANA system at the time you run the plan. From the *Executed Plan* tab, locate the step *Column Union All* and you will see an inclusive time of around 50 ms.
6. How many rows were generated by the union step?

   From the *Executed Plan* tab, locate the *Column Union All* step, and above this step you will see 13,301 rows are generated.

Use the Timeline view

Business Example

You need to identify the longest running steps of your model. To do that, you will use the *Timeline* view.

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.

1. Open the *Timeline* view in the SAP HANA Studio.

2. Using only the *Timeline* view, identify the step occupying most of the plan operator (Pop) time?

Unit 8
Solution 35

Use the Timeline view

Business Example

You need to identify the longest running steps of your model. To do that, you will use the *Timeline* view.

> Note:
> The two letters ## represent the two-digit group number assigned to you by the instructor.

1. Open the *Timeline* view in the SAP HANA Studio.

   a) *Window* → *Show View* → *Other* and expand the node *SAP HANA PlanViz*, select *Timeline*, and choose *OK*.

2. Using only the *Timeline* view, identify the step occupying most of the plan operator (Pop) time?

   Select the longest blue bar in the *Timeline* view and the corresponding step in the flow graph (top part of the screen) will be highlighted. The step is related to the processing of the analytic view CEA1_00 and is where a significant amount of time is spent during execution.

LESSON SUMMARY

You should now be able to:

- Understanding the cost of SQL Statements

Learning Assessment

1. What are the types of engines included by SAP HANA architecture?

   Choose the correct answers.

   - [ ] A Join engine
   - [ ] B Analytic engine
   - [ ] C Calculation engine
   - [ ] D Attribute engine
   - [ ] E OLAP engine

2. Each type of Information View is processed by one SAP HANA Engine

   Determine whether this statement is true or false.

   - [ ] True
   - [ ] False

3. Identify the principles that you need to follow during modeling.

   Choose the correct answers.

   - [ ] A You have to perform calculations before aggregation.
   - [ ] B You should create joins on key columns.
   - [ ] C You have to transfer large sets of data between database and client.
   - [ ] D You have to reduce data transfer between the views.

Unit 8

Learning Assessment - Answers

1. What are the types of engines included by SAP HANA architecture?

   Choose the correct answers.

   - [x] A Join engine
   - [ ] B Analytic engine
   - [x] C Calculation engine
   - [ ] D Attribute engine
   - [x] E OLAP engine

2. Each type of Information View is processed by one SAP HANA Engine

   Determine whether this statement is true or false.

   - [ ] True
   - [x] False

3. Identify the principles that you need to follow during modeling.

   Choose the correct answers.

   - [ ] A You have to perform calculations before aggregation.
   - [x] B You should create joins on key columns.
   - [ ] C You have to transfer large sets of data between database and client.
   - [x] D You have to reduce data transfer between the views.

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com